Document Title	Specification of FlexRay Transport Layer
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	029
Document Classification	Standard

Document Version	2.3.0
Document Status	Final
Part of Release	3.1
Revision	5

Document Change History

Date	Version	Changed by	Change Description
15.09.2010	2.3.0	AUTOSAR Administration	• Added FRTP222, FRTP223 • Modified FRTP195 • Use parameter PduInfoType in callback RxIndication • Legal disclaimer revised
23.06.2008	2.2.1	AUTOSAR Administration	Legal disclaimer revised
17.12.2007	2.2.0	AUTOSAR Administration	• Clarified the role and purpose of the functions PduR_FrTpChangeParameterConfirmation() and PduR_FrTpCancelTransmitConfirmation() with respect to the PDU Router. • Document meta information extended • Small layout adaptations made
24.01.2007	2.1.1	AUTOSAR Administration	• “Advice for users” revised • “Revision Information” added
05.12.2006	2.1.0	AUTOSAR Administration	• Correction in Interaction Diagram • Various descriptions adapted in Chapter 10 • Added BSW00435 due to WP112 decision • Changing API FrTp_Transmit • Several wording corrections • Adaptation of chapter 5.4.2 to new SRS Requirement • Legal disclaimer revised
25.04.2006	2.0.0	AUTOSAR Administration	Document structure adapted to common Release 2.0 SWS Template.
19.09.2005	1.0.0	AUTOSAR Administration	Initial Release

Disclaimer

Advice for users

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

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

Table of Contents

1 Introduction and functional overview

This specification describes the functionality and API of the AUTOSAR basic software: FlexRay Transport Layer (FrTp).

FRTP001: The FrTp Layer is between the PDU Router and the FlexRay Interface module (see Figure 1, according to [2]). This module's main purpose is segmentation and reassembly of messages that do not fit in one of the assigned Fr N-PDUs.

The PDU Router deploys I-PDUs of AUTOSAR COM or DCM to different communication protocols. The routing through a network system type (e.g. CAN, LIN and FlexRay) depends on the I-PDU identifier. The PDU-Router is also in charge of determining whether a transport protocol has to be used or not.
The FlexRay Interface (FrIf) provides equal mechanisms to access a FlexRay bus channel regardless of its location (µC internal/external). It abstracts from the location of FlexRay controllers (on chip / onboard), the ECU hardware layout and the number of FlexRay drivers. The FrIf is in charge to route received PDUs between the FrTp, the PDU Router, the FlexRay NM and the XCP (the latter one does not yet exist, so the position of this module is just an AUTOSAR assumption).

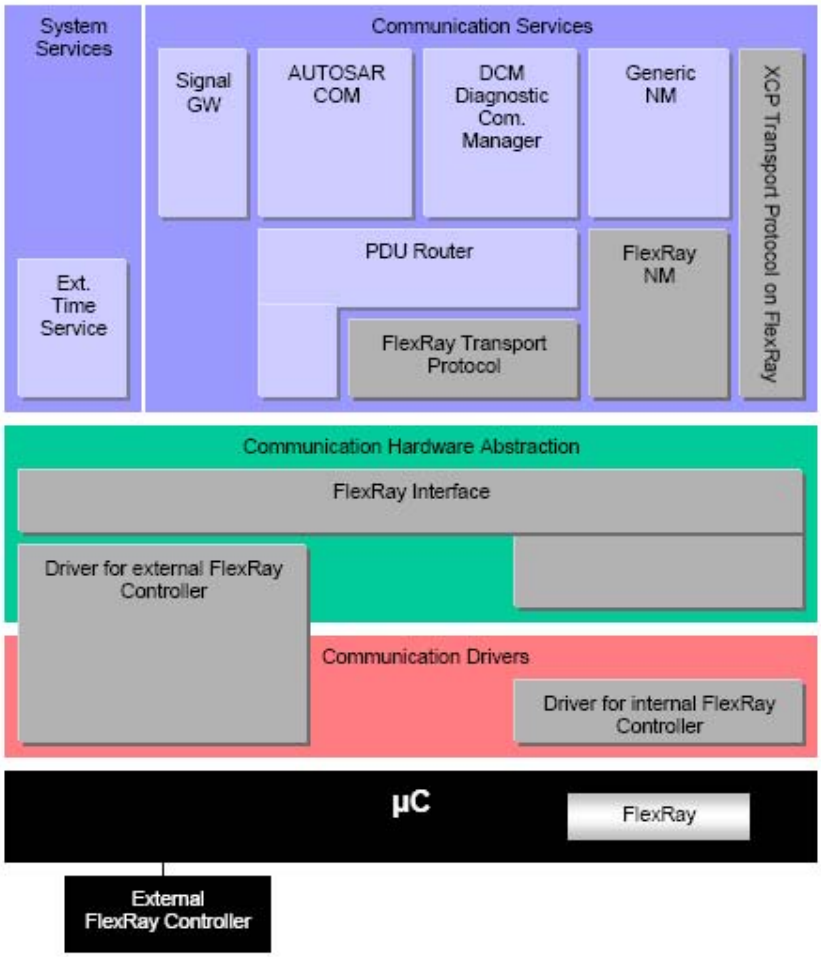


Figure 1: AUTOSAR FlexRay Layered Architecture

FRTP002: Among others, the FlexRay Transport Layer includes the following features:

- Segmentation of data in send direction
- Collection of data in receive direction
- Control of data flow
- Detection of errors
- Acknowledgement (and Retry)
- 1:1 and 1:n connections
- 2 or 4 Bytes address information
- Transfer of up to 2^{32}-1 Bytes payload
- Configurable to be compliant to both ISO 15765-2 and ISO 15765-4

FRTP003: It is an AUTOSAR decision to base on existing standards the specification of basic software module. So this AUTOSAR FlexRay Transport Layer specification is based on the international standard ISO 15765 which are the most used in automotive area.

FRTP004: The basic idea is, to have an ISO 15765-2 and ISO 15765-4 compliant Transport Layer, which allows by the means of static configuration to add one or more optional features (like acknowledgement) per channel independently of each other. Of course, by adding such a feature the compliance to these both ISO specifications gets lost for this particular channel.
Additionally, the features are deactivateable at compile time. But if they are compiled in, they are still deacitvateable by static configuration.
The rationale behind some of the provided features is the usage of this transport layer not only for diagnostic purposes but also for Inter-ECU communication.

FRTP005: Since addressing within ISO 15765-2 is specific for the CAN bus system (CAN identifier), it is obvious that another approach is taken within FlexRay Transport Layer.

FRTP006: Although FlexRay transport protocol is at first set to vehicle diagnostic systems, it has been developed to also deal with requirements from other FlexRay based systems needing a transport layer protocol.

2 Acronyms and abbreviations

Following acronyms and abbreviations have a local scope only and therefore are not contained in the AUTOSAR glossary.

Acronym:	*Description:*
Channel	A group of connections sharing the properties configurable by the parameters in chapter 10.4
Connection	Way of communication between sender / receiver, characterized by the parameters in chapters 10.4. Uniquely identified by the parameter `FRTP_SDUID`.
Frame	Synonym for Fr N-PDU → One TP Frame cannot be split up into several Fr N-PDUs
Fr L-SDU	This is the SDU of the FlexRay Interface module. It represents the same entity as Fr N-PDU but with the FlexRay Interface module's point of view.
Fr L-SduId	Unique identifier of a SDU within the FlexRay Interface. It is used for referencing Fr L-SDU's routing properties. Consequently, to interact with the FlexRay Interface via its API, an upper layer uses Fr LSduId to refer to an Fr L-SDU Info Structure.
Fr N-PDU	This is a PDU of the FlexRay Transport Layer, which is given to the FlexRay Interface for Sending. It consists of address information, protocol control information and the payload (Fr N-SDU).
Fr N-SDU	This is the SDU of the FlexRay Transport layer. In the AUTOSAR architecture, it is a set of data coming from the PDU Router. Each FR N-SDU is connected to a unique identifier.
Fr N-SDU Info Structure	This is a FlexRay Transport Layer internal constant structure that contains specific FlexRay Transport Layer information to process transmission, reception, segmentation and reassembly of the related Fr N-SDU.
Fr N-SduId	Unique identifier of a SDU within the FlexRay Transport Layer. It is used for referencing FR N-SDU's routing properties. Consequently, to interact with the FlexRay Transport Layer via its API, an upper layer uses Fr NSduId to refer to an Fr N-SDU Info Structure.
I-PDU	This is the PDU of the AUTOSAR COM module
Message	Synonym for Fr N-SDU
PDU	In layered systems, it refers to a unit of data that is specified in a protocol of a given layer and that consists of user data of that layer (SDU) plus possibly protocol control information of that given layer. In fact, the PDU of layer X is the SDU of its lower layer X-1 (i.e. (X)-PDU = (X-1)-SDU).
SDU	In layered systems, it refers to a set of data that is sent by a user of the services of a given layer, and is transmitted to a peer service user semantically unchanged.

Abbreviation:	*Description:*
AF	Acknowledgement Frame Fr N-PDU
CF	Consecutive Frame Fr N-PDU
Com	AUTOSAR COM module
Dcm	Diagnostic Communication Manager module
FC	Flow Control Fr N-PDU
FF	First Frame Fr N-PDU
Fr	FlexRay Driver module
Fr N-PCI	Protocol Control Information of the transport layer
FrIf	FlexRay Interface
FrTp	FlexRay Transport Layer
N_AI	Network Address Information
PDU	Protocol Data Unit
PduR	PDU Router

SDU	Service Data Unit
SF	Single Frame Fr N-PDU
XCP	X (CAN, FlexRay, …) Calibration Protocol

3 Related documentation

3.1 Input documents

[1] List of Basic Software Modules
AUTOSAR_BasicSoftwareModules.pdf

[2] Layered Software Architecture
UOSAR_LayeredSoftwareArchitecture.pdf

[3] General Requirements of Basic Software Modules
AUTOSAR_SRS_General.pdf

[4] Requirements on FlexRay
AUTOSAR_SRS_FlexRay.pdf

[5] Specification of FlexRay Interface
AUTOSAR_SWS_FlexRay_Interface.pdf

[6] Specification of Communication Stack Types
AUTOSAR_SWS_ComStackTypes.pdf

[7] Specification of Standard Types
AUTOSAR_SWS_StandardTypes.pdf

[8] Specification of Platform Types
AUTOSAR_SWS_PlatformTypes.pdf

[9] AUTOSAR Basic Software Module Description Template,
AUTOSAR_BSW_Module_Description.pdf

3.2 Related standards and norms

[10] ISO 15765-2(2003-11-11), Road vehicles — Diagnostics on Controller Area Networks (CAN) — Part2: Network layer services

[11] ISO 15765-4(2004-09-07), Road vehicles — Diagnostics on Controller Area Networks (CAN) — Part4: Requirements for emissions-related systems

[12] FlexRay Communications System Protocol Specification Version 2.1

4 Constraints and assumptions

4.1 Limitations

AUTOSAR architecture defines protocol specific transport layer (CanTp, LinTp, FrTp…). So the FlexRay Transport Layer covers only FlexRay transport protocol specifics.

The FlexRay Transport Layer has an interface to a single underlying FlexRay Interface Layer and a single upper PDU Router module.

At least, the first version of AUTOSAR will not support transport protocol facilities for AUTOSAR COM. Therefore, non-diagnostic I-PDUs are limited to the configured payload size of the FlexRay bus.

4.2 Applicability to car domains

The FlexRay Transport Layer can always be used for applications if the FlexRay protocol was used.

5 Dependencies to other modules

This section sets out relations between the FrTp and other AUTOSAR basic software modules. It contains brief descriptions of the services, which are required by the FrTp from other modules or other modules can call at the FrTp. The following picture gives a brief overview of the interactions.

FRTP007:

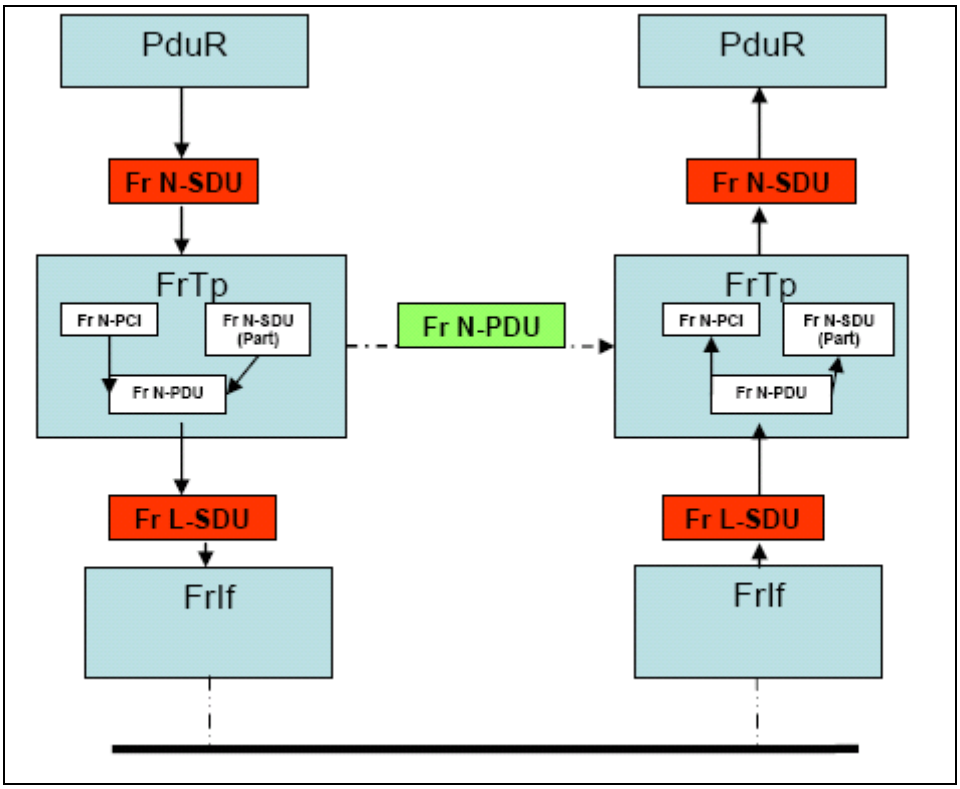


Figure 2: FrTp interactions

5.1 PduRouter

The following services of the PduRouter are called by the FrTp:

- *PduR_FrTpRxIndication*
  By this API service, the FrTp indicates the completed (un)successful reception of a message
- *PduR_FrTpProvideRxBuffer*
  By this API service, the FrTp asks the actual receiver (e. g. DCM) of the message to provide a receive buffer. It is not necessary for the buffer to have at least the same size as the whole Fr N-SDU length (there will be another call in this case).

- *PduR_FrTpProvideTxBuffer*
  By this API service, the FrTp asks the actual sender (e.g. DCM) of the message to provide a transmit buffer. It is not necessary for the buffer to have at least the same size as the whole Fr N-SDU length (there will be another call in this case) but a minimum length (due to FrTp internal exigencies) will be passed.
- *PduR_FrTpTxConfirmation*
  By this API service, the FrTp confirms the (un)successful sending of the complete message (Fr N-SDU) to the actual sender (e. g. DCM).
- *PduR_FrTpChangeParameterConfirmation*
  By this API service, the FrTp confirms the (un)successful execution of *FrTp_ChangeParameterRequest.*
- *PduR_FrTpCancelTransmitConfirmation*
  By this API service, the FrTp confirms the (un)successful execution of *FrTp_CancelTransmitRequest.*

The following services of the FrTp are called by the PduRouter:

- *FrTp_Transmit*
  By this API service, the sending of a message (Fr N-SDU) is triggered. The FrTp will then ask for a transmit buffer and start sending.
- *FrTp_CancelTransmitRequest*
  By this API service, the sending or receiving of a message (Fr N-SDU) is cancelled. This service is optional (per channel).
- *FrTp_ChangeParameterRequest*
  By this API service, some parameters of a channel can be changed. This service is optional (per channel).

5.2 FlexRay Interface

The following services of the FlexRay Interface are called by the FrTp:

- *FrIf_Transmit*
  By this API service, the sending of a message (Fr N-PDU) is triggered. Depending on configuration on the FlexRay Interface, the Fr N-PDU is sent immediately or after the call of FrTp_TriggerTransmit.

The following services of the FrTp are called by the FlexRay Interface:

- *FrTp_RxIndication*
  By this API service, the FlexRay Interface indicates the reception of an FrTp frame (Fr N-PDU, please do not mistake this with a FlexRay frame) to the FrTp. The FrTp then processes this frame.

- *FrTp_TxConfirmation*
  By this API service, the FlexRay Interface confirms the sending of the frame containing the Fr N-PDU over the FlexRay network.
- *FrTp_TriggerTransmit*
  By this API service, the FlexRay Interface makes the FrTp to copy the Fr N-PDU into the buffer provided by the FlexRay Interface. The FlexRay interface then can start sending the FlexRay frame containing the Fr N-PDU.

5.3 COM Manager

The following services of the FrTp are called by the COM Manager:

- *FrTp_Init*
  By this API service, all global variables are initialized and each connection is set into the Idle state.
- *FrTp_Shutdown*
  By this API service, all pending transport connections are closed, resources are freed and the module is stopped.

5.4 File structure

5.4.1 Code file structure

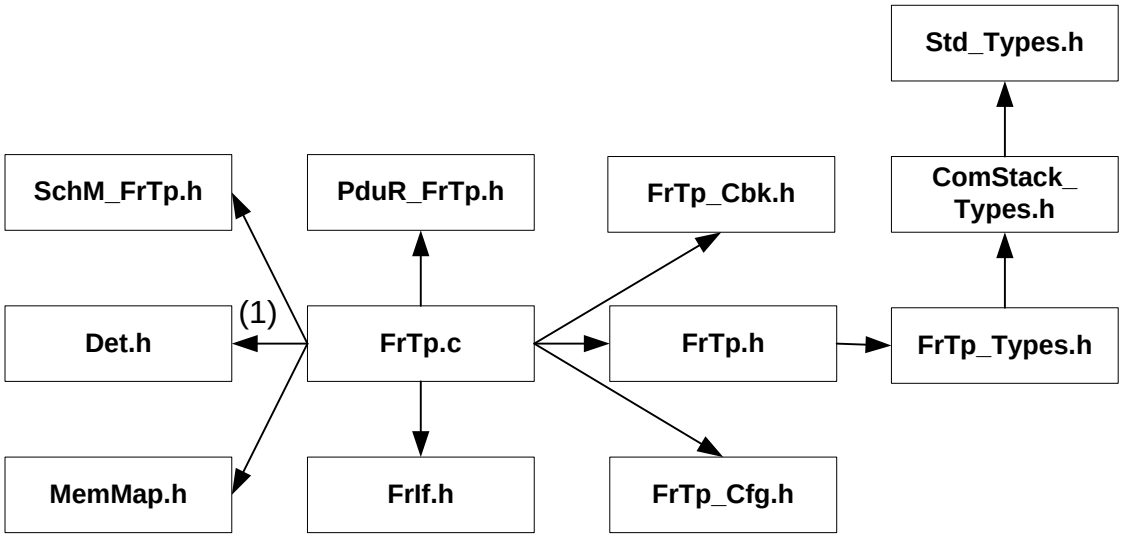
FRTP214: The Code file structure shall include the following files named:
- FrTp.c – the source code
- FrTp_Lcfg.c – for link time configurable parameters and
- FrTp_PBcfg.c – for post build time configurable parameters.

These files shall contain all link time and post-build time configurable parameters.

5.4.2 Header file structure

FRTP195: The Header file structure shall include the following files named:
- FrTp.h - general header file
- FrTp_Cfg.h - pre-compile time configuration parameters
- Det.h – header file of Det
- PduR_FrTp.h – header file of PduR
- FrIf.h – header file of FrIf
- SchM_FrTp.h – header file of TP related SchM declarations
- MemMap.h – header file for Memory Mapping
- Std_Types.h – header file for standard types
- ComStack_Types.h – header file for ComStack types
- FrTp_Types.h – header file for FrTP specific types

FRTP222: The FrTp.h file shall include FrTp_Types.h



(1) … only if development error detection is enabled

5.4.3 Design Rules

FRTP208: The code of the Fr Transport Protocol (as long as it is written in C) shall conform to the HIS subset of the MISRA C Standard.

FRTP209: Direct use of compiler and platform specific keywords shall be avoided.

FRTP210: Indicate all global data with read-only purposes by explicitly assigning the `const` keyword.

FRTP211: It is allowed to use macros instead of functions where source code is used and runtime is critical.

FRTP212: No global data shall be defined in the header files. If global variables have to be used, the definition shall take place in the C file.

FRTP213: The source code of the Fr Transport Protocol module shall not be processor and compiler dependent.

6 Requirements traceability

Document: General Requirements of Basic Software Modules

Requirement	***Satisfied by***
[BSW00160] Human-readable configuration data	Fulfilled by configuration chapter
[BSW00161] Microcontroller abstraction	not applicable
[BSW00162] ECU layout abstraction	not applicable
[BSW00172] Compatibility and documentation of scheduling strategy	not applicable
[BSW003] Version identification	FRTP200:
[BSW003] Version identification	FRTP178:
[BSW00300] Module naming convention	Fulfilled by API definitions in chapter 8
[BSW00301] Limit imported information	not applicable
[BSW00302] Limit exported information	not applicable
[BSW00304] AUTOSAR integer data types	Fulfilled by API definitions in chapter 8
[BSW00305] Self-defined data types naming convention	Fulfilled by type definitions in chapter 8
[BSW00306] Avoid direct use of compiler and platform specific keywords	FRTP209:
[BSW00307] Global variables naming convention	not applicable
[BSW00308] Definition of global data	FRTP212:
[BSW00309] Global data with read-only constraint	FRTP210:
[BSW00310] API naming convention	FRTP207:
[BSW00312] Shared code shall be reentrant	Here the means are described so this is a requirement to implementation
[BSW00314] Separation of interrupt frames and service routines	not applicable
[BSW00318] Format of module version numbers	FRTP178:
[BSW00321] Enumeration of module version numbers	not applicable
[BSW00323] API parameter checking	FRTP205:
[BSW00324] Do not use HIS I/O Library	not applicable
[BSW00325] Runtime of interrupt service routines	not applicable
[BSW00326] Transition from ISRs to OS tasks	not applicable
[BSW00327] Error values naming convention	Fulfilled by chapter 7.7
[BSW00328] Avoid duplication of code	This is a requirement to implementation
[BSW00329] Avoidance of generic interfaces	not applicable
[BSW00330] Usage of macros / inline functions instead of functions	FRTP211:
[BSW00331] Separation of error and status values	Fulfilled by the different types
[BSW00333] Documentation of callback function context	Fulfilled by API definitions in chapter 8
[BSW00334] Provision of XML file	not applicable
[BSW00335] Status values naming convention	not applicable
[BSW00336] Shutdown interface	FRTP148:
[BSW00337] Classification of errors	FRTP179:
[BSW00338] Detection and Reporting of development errors	FRTP177:
[BSW00339] Reporting of production relevant errors and exceptions	not applicable (No productions are available)
[BSW00341] Microcontroller compatibility documentation	not applicable
[BSW00342] Usage of source code and object code	not applicable

[BSW00343] Specification and configuration of time	Fulfilled by configuration chapter
[BSW00344] Reference to link-time configuration	not applicable (no link-time only parameters)
[BSW00345] Configuration at Compile time	FRTP177: , FRTP178:
[BSW00346] Basic set of module files	FRTP195:
[BSW00347] Naming separation of different instances of BSW drivers	not applicable For driver only.
[BSW00348] Standard type header	not applicable
[BSW00350] Development error detection keyword	FRTP177:
[BSW00353] Platform specific type header	not applicable
[BSW00355] Do not redefine AUTOSAR integer data types	Fulfilled by API definitions in chapter 8
[BSW00357] Standard API return type	Fulfilled by API definitions in chapter 8
[BSW00358] Return type of `init()` functions	Fulfilled by API definitions in chapter 8
[BSW00359] Return type of callback functions	Fulfilled by API definitions in chapter 8
[BSW00360] Parameters of callback functions	Fulfilled by API definitions in chapter 8
[BSW00361] Compiler specific language extension header	not applicable
[BSW00369] Do not return development error codes via API	FRTP149: - FRTP154:
[BSW00370] Separation of callback interface from API	Fulfilled by chapter 8
[BSW00371] Do not pass function pointers via API	Fulfilled by API definitions in chapter 8
[BSW00373] Main processing function naming convention	Fulfilled by API definitions in chapter 8
[BSW00374] Module vendor identification	FRTP178:
[BSW00375] Notification of wake-up reason	not applicable
[BSW00376] Return type and parameters of main processing functions	Fulfilled by API definitions in chapter 8
[BSW00377] Module specific API return types	Fulfilled by API definitions in chapter 8
[BSW00378] AUTOSAR boolean type	Fulfilled by API definitions in chapter 8
[BSW00379] Module identification	FRTP178:
[BSW00380] Separate C-Files for configuration parameters	FRTP166:
[BSW00381] Separate configuration header file for pre-compile time parameters	FRTP195: FRTP214:
[BSW00382] Not-used configuration elements need to be listed	not applicable
[BSW00383] List dependencies of configuration files	FRTP195: FRTP214:
[BSW00384] List dependencies to other modules	FRTP007:
[BSW00385] List possible error notifications	Fulfilled by chapter 7.7
[BSW00386] Configuration for detecting an error	Fulfilled by chapter 10
[BSW00387] Specify the configuration class of callback function	Fulfilled by chapter 8
[BSW00388] Introduce containers	Fulfilled by chapter 10.2
[BSW00389] Containers shall have names	Template requests names, so the requirement is fulfilled
[BSW00390] Parameter content shall be unique within the module	Parameters are unique
[BSW00391] Parameter shall have unique names	Parameters have unique names
[BSW00392] Parameters shall have a type	Template requests type, so the requirement is fulfilled
[BSW00393] Parameters shall have a range	Template requests range, so the requirement is fulfilled
[BSW00394] Specify the scope of the parameters	Template requests scope, so the requirement is fulfilled

[BSW00395] List the required parameters (per parameter)	not applicable
[BSW00396] Configuration classes	Parameter-template requests configuration classes
[BSW00397] Pre-compile-time parameters	This is not a requirement, it is a description
[BSW00398] Link-time parameters	This is not a requirement, it is a description
[BSW00399] Loadable Post-build time parameters	Done by configuration description
[BSW004] Version check	FRTP201:
[BSW00400] Selectable Post-build time parameters	not applicable
[BSW00401] Documentation of multiple instances of configuration parameters	Fulfilled by configuration chapter
[BSW00402] Published information	FRTP178:
[BSW00404] Reference to post build time configuration	FRTP195:
[BSW00405] Reference to multiple configuration sets	not applicable
[BSW00406] Check module initialization	To perform this check the start up code of the microcontroller has to initialize the status variables. Furthermore E_UNINIT is not defined in the Std_ReturnType
[BSW00407] Function to read out published parameters	FRTP202:
[BSW00408] Configuration parameter naming convention	Fulfilled by chapter 10
[BSW00409] Header file for production error code IDs	not applicable
[BSW00410] Compiler switch shall have defined values	Template requests compiler switches with defined values, so the requirement is fulfilled
[BSW00411] Get version info keyword	FRTP215:
[BSW00412] Separate H-File for configuration parameters	not applicable, post build time configuration is referenced in the init-function
[BSW00413] Accessing instances of BSW modules	not applicable Only 1 instance of FrTp allowed.
[BSW00414] Parameter of init function	Fulfilled by API definitions in chapter 8
[BSW00415] User dependent include files	not applicable
[BSW00416] Sequence of initialization	not applicable
[BSW00417] Reporting of Error Events by Non-Basic Software	not applicable
[BSW00419] Separate C-Files for pre-compile time configuration parameters	not applicable
[BSW00420] Production relevant error event rate detection	not applicable DEM requirement
[BSW00421] Reporting of production relevant error events	FRTP206:
[BSW00422] Debouncing of production relevant error status	not applicable DEM requirement
[BSW00423] Usage of SW-C template to describe BSW modules with AUTOSAR Interfaces	Template used
[BSW00424] BSW main processing function task allocation	FRTP203:
[BSW00425] Trigger conditions for schedulable objects	not applicable
[BSW00426] Exclusive areas in BSW modules	not applicable
[BSW00427] ISR description for BSW modules	not applicable No ISR function
[BSW00428] Execution order dependencies of main processing functions	not applicable FlexRay TP has only one MainFunction

[BSW00429] Restricted BSW OS functionality access	not applicable
[BSW00431] The BSW Scheduler module implements task bodies	not applicable
[BSW00432] Modules should have separate main processing functions for read/receive and write/transmit data path	Fulfilled by chapter 8
[BSW00433] Calling of main processing functions	not applicable
[BSW00434] The Schedule module shall provide an API for exclusive areas	not applicable
[BSW00435] Module Header File Structure for the Basic Software Scheduler	FRTP195:
[BSW00436] Module Header File Structure for Memory Mapping	FRTP195:
[BSW005] No hard coded horizontal interfaces within MCAL	not applicable
[BSW006] Platform independency	FRTP213:
[BSW007] HIS MISRA C	FRTP208:
[BSW009] Module User Documentation	Fulfilled by the whole document
[BSW010] Memory resource documentation	not applicable
[BSW101] Initialization interface	FRTP147:
[BSW158] Separation of configuration from implementation	Redundant to BSW00346
[BSW159] Automatic configuration	FRTP168: - FRTP178: , FRTP180: , FRTP181:
[BSW164] Implementation of interrupt service routines	not applicable
[BSW167] Static configuration checking	FRTP171: FRTP174: , FRTP180: , FRTP181:
[BSW168] Diagnostic interface	not applicable
[BSW170] Data for reconfiguration of SW-components	not applicable
[BSW171] Configurability of optional functionality	FRTP168:
[BSW375] Notification of wake-up reason	not applicable

Document: AUTOSAR requirements on Basic Software, cluster FlexRay

Requirement	***Satisfied by***
BSW05073 (Usage of ISO 15765-2 and ISO 15765-4 specifications)	FRTP003: , FRTP004: , FRTP005:
BSW05074 (FlexRay Transport Interfaces)	FRTP001:
BSW05075 (Configuration Independence)	FRTP001: , FRTP149:
BSW05123 (Configuration Modifiable by a Flashing Process)	FRTP166:
BSW05076 (Multiple Logical FlexRay Transport Layer Channels)	FRTP088: , FRTP089:
BSW05077 (Unique Identifier of N-SDU)	FRTP168:
BSW05079 (Transport Connection Properties)	FRTP168:
BSW05124 (Global Transport Layer Properties)	FRTP177:
BSW05082 (Acknowledgement without Retry)	FRTP168: , FRTP012: , FRTP013:
BSW05083 (Acknowledgement with Retry)	FRTP168: , FRTP014: , FRTP015:
BSW05085 (Segmented 1:n Connections without Flow Control)	FRTP168: , FRTP016: , FRTP017:
BSW05104 (Default Separation Time)	FRTP168:
BSW05088 (FlexRay Transport Layer Initialization)	FRTP147:
BSW05089 (FlexRay Transport Layer Availability)	FRTP179:
BSW05090 (Support of Optional ISO 15765-2 Service)	FRTP104:

BSW05093 (Transmit Cancellation)	FRTP099:- FRTP103:
BSW05095 (Bandwidth Control)	FRTP011:
BSW05129 (Mismatch of Service Call and Connection Properties)	FRTP149: - FRTP151:

7 Functional specification

FRTP008: The FrTp offers services for segmentation, transmission with flow control, and reassembly of messages (Fr N-SDUs). Its main purpose is to transfer messages that may or may not fit in a single FlexRay frame.

FRTP201: The FlexRay TP shall perform a preprocessor-check if its source and header files belong to the same version.

FRTP200: A readable software version number shall be included in the header file.

7.1 Overview

FRTP009: Beside the features according to ISO 15765-2 (7 byte data per frame, 4 kByte message length, unsegmented 1:n connections, multiple logical channels concurrently, flow control, service request confirmation) it allows to configure independently of each other the following features for a specific channel at both pre- and post-compile time:

- Acknowledgement (with or without Retry) for 1:1 connections
- Segmented 1:n connections (without flow control)
- Transmission cancellation
- Up to 2^{32}-1 Byte message length

Additionally the length and the number of the Fr N-PDUs of a connection is configurable. It has to be clear, that the N-PDUs are unique for a channel and for each connection of a channel only N-PDUs having the same length can be chosen.

For the rest of this document, sections or features which are not compliant to the above mentioned ISO standard will be marked as "**Not compliant to ISO**" and, if applicable, the corresponding text is written between { }.

7.2 Protocol Processes

There are, as will be shown later on, different types of First Frames and Single Frames. So in the sequence diagrams will always FF or SF be used, regardless of the concrete subtype.

7.2.1 1:1 Connections

This type of connection is the most common in today's automotive applications. Within the FlexRay Transport Layer the following subtypes are possible.

7.2.1.1 1:1 Connection in a channel without Acknowledgement

FRTP010: Unsegmented Transfer
In case a message does not exceed the possible amount of payload for a SF (which can be derived from the values of FRTP_PDU_LENGTH, FRTP_ADRTYPE and FRTP_LM), there is no need to segment this message.

The transfer takes place as illustrated in Figure 3:

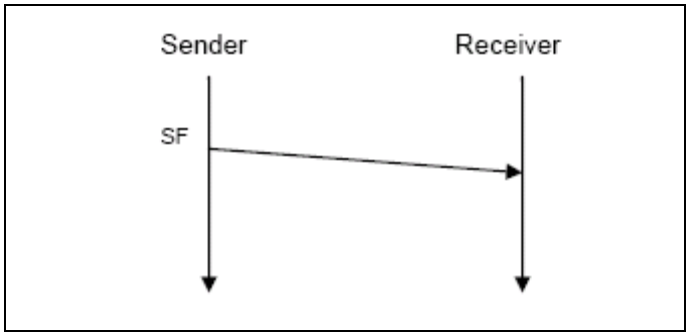


Figure 3: Unsegmented 1:1 transfer without acknowledgement

The sending Transport Layer packs the payload (Fr N-SDU) into an Fr N-PDU and sends it to the receiving Transport Layer. This is done via a Single Frame (SF).

FRTP011: Segmented Transfer
In case a message does not fit into an SF, it needs to be split up into several parts and flow control is applied to control the data flow taking into account the needs of the receiver.

In this case, the transfer takes place as shown in Figure 4:

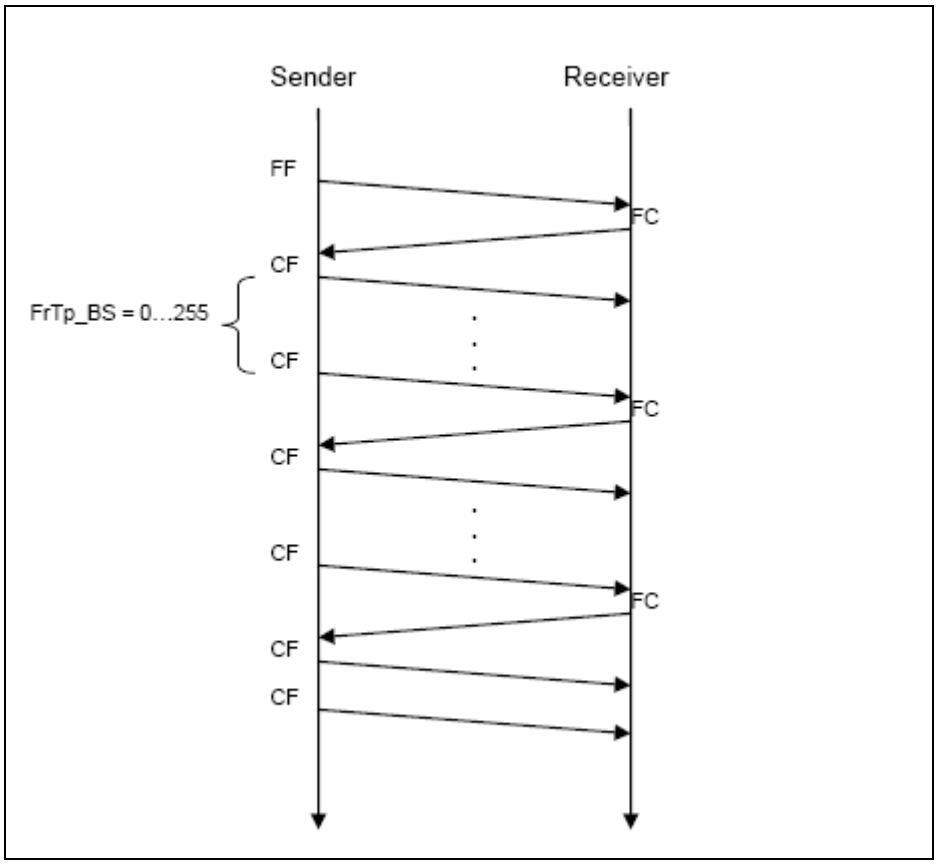


Figure 4: Segmented 1:1 transfer without acknowledgement

The transfer starts with sending a First Frame (FF) from the sender to the receiver. This frame contains the length of the whole message (e. g. 1000 Byte) and even the first data bytes.
The receiving peer reacts to the reception of a FF with sending of a Flow Control frame (FC) back to the sender. This FC frame contains the value of three parameters: FRTP_FS, FRTP_BS and FRTP_STMIN.

FRTP_FS states the flow status. The possible values are:

- CTS: Clear To Send
  The sender can continue transmitting the message
- WT: Wait
  The sender shall wait for another FC frame.
- OVFLW: Overflow
  The sender shall abort the transfer, because the receiver has not enough buffer for the whole message available.

There shall be a statically defined upper limit (FRTP_MAX_BUFREQ) for the number of allowed WT's. If this number has been reached, the transmission shall be aborted and within *PduR_FrTpTxConfirmation* the result FRTP_WT_OVRN shall be returned.

FRTP_BS specifies the block size. This is the number of Consecutive Frames (CF) the sender is allowed to send between two FC Frames. The possible range is from 0x00 to 0xFF, whereas 0x00 states that no more FC Frames will be transmitted by the receiver, i. e. the whole message shall be sent in one big block.

FRTP_STMIN quotes the minimum gap between two CFs in milliseconds or microseconds. The range from 0x00 to 0x7F specifies the minimum gap in milliseconds, the one from 0xF1 to 0xF9 defines the gap in microseconds (100 µs, 200 µs, …)

The alternating transmission of CF blocks and a FC frame lasts, until the whole message is sent.

The FRTP_STMIN parameters can be changed during runtime by using the respective API call.

7.2.1.2 1:1 Connection in a channel with Acknowledgement without Retry

This subchapter is **Not compliant to ISO** and describes how a simple acknowledgement mechanism looks like.

FRTP012: Unsegmented Transfer
This is mostly done like in section Unsegmented Transfer of chapter 7.2.1.1, except that there is an additional Acknowledge Frame (AF) which is sent from the receiver to the sender. This is illustrated in Figure 5:

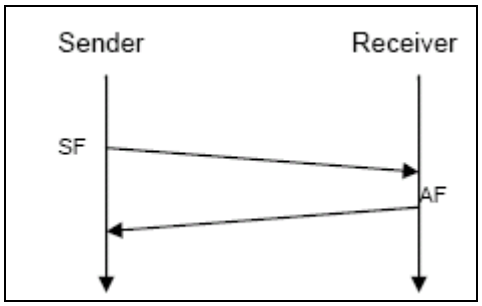


Figure 5: Unsegmented 1:1 transfer with Acknowledgement without Retry

The AF contains among others the parameter FRTP_ACK which has two possible values, Positive Acknowledgement (POS_ACK) or Negative Acknowledgement (NEG_ACK). Thus the sender is informed about the (un)successful reception of a message by the receiving peer. If the FS field of an AF frame (see chapter 7.3.6) contains the value WT, another AF, up to FRTP_MAX_BUFREQ, will arrive.

FRTP013: Segmented Transfer
This is done very similar to section Segmented Transfer of chapter 7.2.1.1. There are only three differences:

The first difference is the transmission of an AF after the last block, because this one has to be acknowledged as well. This frame is similar to an ordinary Flow Control frame but contains additionally the FRTP_ACK parameter (for positive or negative acknowledgement) and the sequence number of the first faulty frame of the transmitted block.
The second difference is the transmission of an AF with a negative acknowledgement after a block in which an error occurred. This AF also contains the sequence number of the first faulty or missing frame.
The third difference is, that the block size shall be in the range from 1 to 16 (due to the 4 bit sequence number, see chapter 7.3.4)

The procedure can be seen in Figure 6:

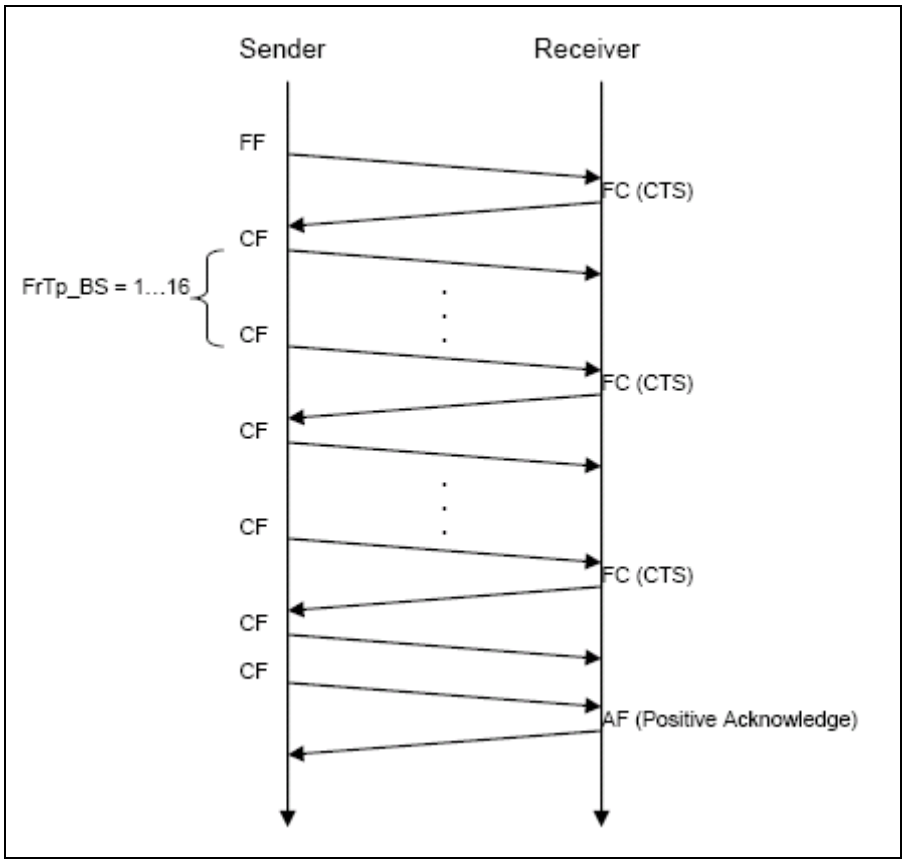


Figure 6: Segmented 1:1 transfer with Acknowledgement without Retry

Obviously, the acknowledgement is done on a “per block” basis, depending on the current block size.
In case of a negative acknowledgement after a block (in that case instead of an FC frame an AF with a negative acknowledgement is sent to the sender and the receiver aborts the reception and indicates an appropriate result to its upper layer (*PduR_FrTpRxIndication*) the sender aborts the transmission and informs its upper layer (*PduR_FrTpTxConfirmation*).

7.2.1.3 1:1 Connection in a channel with Acknowledgement with Retry

This subchapter is **Not compliant to ISO**

FRTP014: Unsegmented Transfer
This section is quite similar to the corresponding one in chapter 7.2.1.2. The only difference is that in case of a negative acknowledgement the frame is retransmitted.

This behaviour is depicted in Figure 7 and Figure 8:

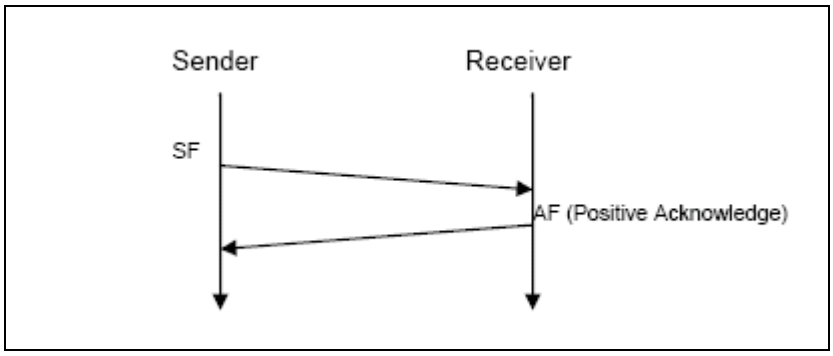


Figure 7: Unsegmented 1:1 transfer, Acknowledgement with Retry configured, Positive Acknowledgement

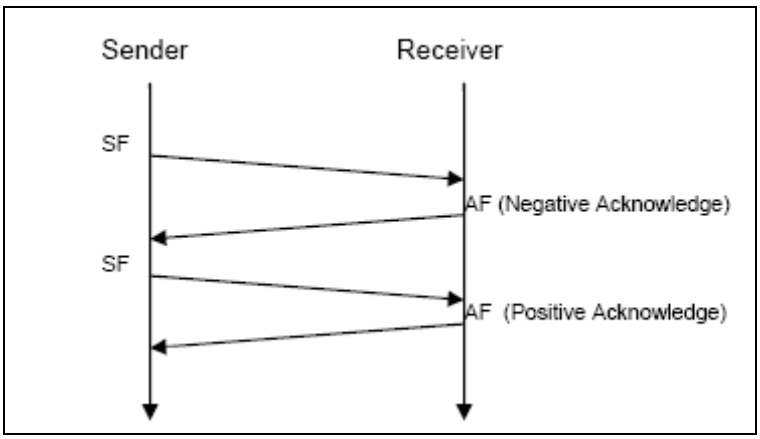


Figure 8: Unsegmented 1:1 transfer, Acknowledgement with Retry configured, Negative Acknowledgement

If in Figure 8 the second try of sending the message also failed, there would be a third one and so on.
In order to prevent infinite retransmissions in the case of a permanent failure, an upper limit (FRTP_MAX_RN) has to be defined. If the number of retries has reached this value, the transmission of the corresponding message shall be stopped and within *PduR_FrTpTxConfirmation* and *PduR_FrTpRxIndication* an adequate result (see chapter 8.2.1) shall be returned.

FRTP015: Segmented Transfer
Compared to the segmented transfer in chapter 7.2.1.2, the difference is the Retry mechanism and ,coming with it, the alternating block mechanism.

The Retry mechanism works as follows:

In the case a negative acknowledge arrives at the sender, this also contains the sequence number of the first faulty frame in the currently transmitted block. Now the sender transmits, starting with the stated sequence number, all remaining frames of the just transmitted block again.
In order to prevent infinite retransmissions in case of a permanent failure, the mentioned parameter limit the retry attempts.
The Retry mechanism is shown in Figure 9 for the case of a block size of 4:

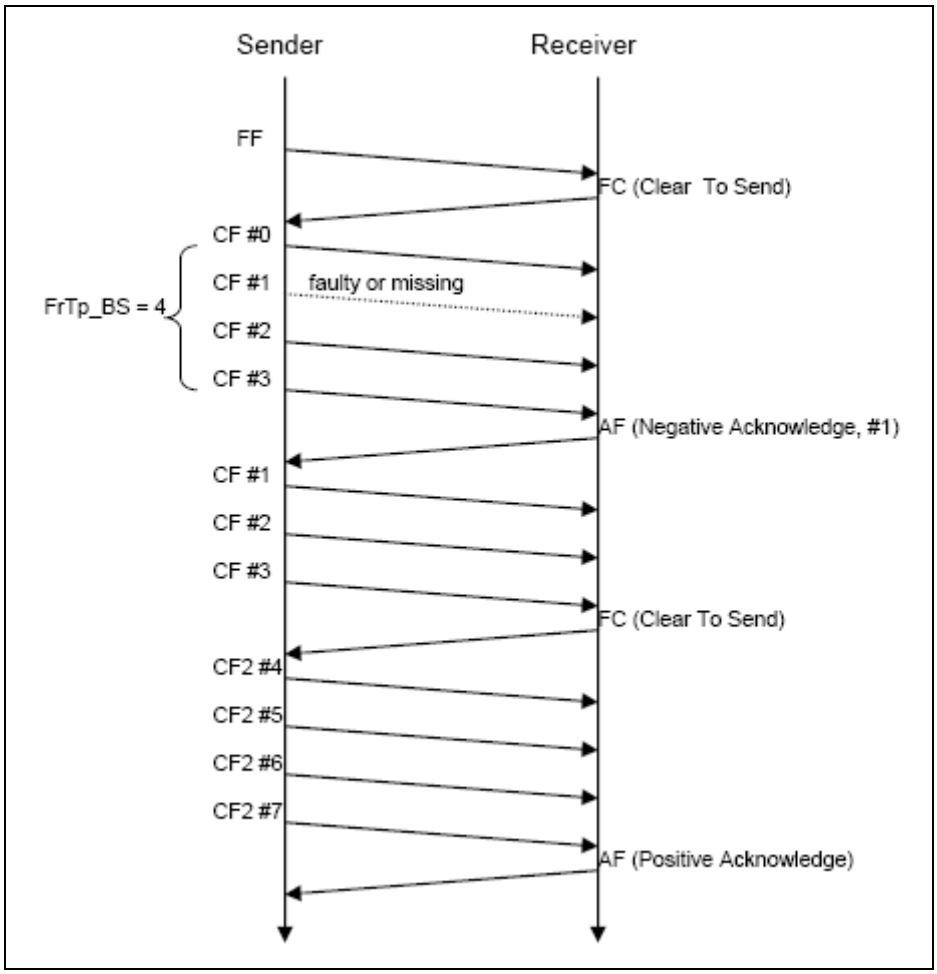


Figure 9: Segmented 1:1 transfer with Acknowledgement with Retry

If the retry starts with a lower sequence number than requested, this shall be tolerated, i. e. all frames until the requested shall be ignored and errors within the ignored frames shall be ignored, too. If it starts with a higher number than requested, this shall lead to another negative acknowledgement after the block end.

Alternating Block Mechanism

When using the Retry mechanism the FlexRay TP transfers blocks using the Alternating Block Mechanism. This works as follows:
The first block is transferred using normal CF frames. The second block is transferred using CF2 frames, the third one with CF frames and so on. When a retry occurs, a CF block is again transferred with CF frames and, of course, a CF2 block is retried with CF2 frames.

This mechanism ensures correct behaviour in case at the block end an FC frame is lost, especially if it is an FC with flow status CTS, by allowing the detection of the unneccessary retries.

7.2.2 1:n Connections

In the case of 1:n connections (1 sender, multiple receivers) there is no further distinction in subtypes (with or without acknowledgement). The reason for this is that the size of the receiving group is often not known a priori, so it is not possible to apply flow control or acknowledgement mechanisms to 1:n connections.
So the only distinction made is between unsegmented and segmented transfer.

FRTP016: Unsegmented Transfer
This is exactly the same like in the section Unsegmented Transfer of chapter 7.2.1.1. The only difference is the multiple receivers instead of one. So the procedure looks like the following:

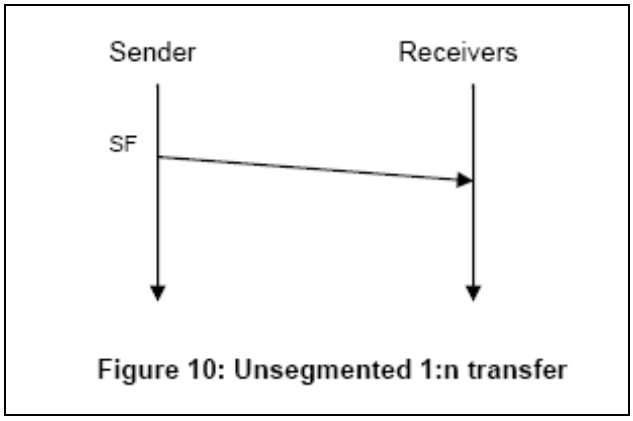


Figure 10: Unsegmented 1:n transfer

One sender sends its message to a group of receivers.

FRTP017: Segmented Transfer Not compliant to ISO
Since no flow control or acknowledgement is possible in this case, a segmented 1:n transfer only consists of a FF and the number of necessary CFs

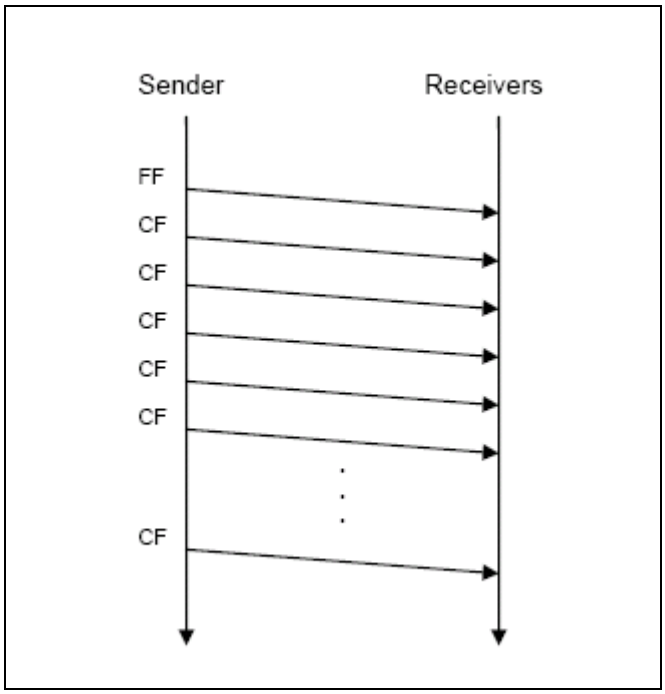


Figure 11: Segmented 1:n transfer

In case an error occurs, the reception will be terminated and the appropriate result will be given within *PduR_FrTpRxIndication()*.

7.3 Frame Layout

As seen in chapter 7.2 there are different types of frames. A detailed explanation of all the types follows below.

7.3.1 General

FRTP018: The general structure of a frame is shown in Figure 12:

Figure 12: Structure of a FlexRay Transport Layer frame

It is common to all frames that they are headed by address information. Depending on static configuration (per channel), in a way whether 1 Byte or 2 Byte addressing is used, this address information consists of 1 Byte for Target Address and 1 Byte for Source Address or 2 Bytes for Target address and 2 Bytes for Source Address. Since it depends on the interpretation of the address information, it is not further

specified whether this address information is utilized for the in automotive area so called “Physical” or for “Functional” addressing.

Although in the following it is talked about frames, it must be clear, that these are NOT frames on the FlexRay physical layer but frames within the FlexRay Transport Layer’s point of view. From FlexRay Interface’s and below view these frames are just PDUs, so an FrTp frame is an Fr N-PDU. The mapping of the following Transport Layer frames in FlexRay Physical Layer frames, takes place within the FlexRay Interface.

Please note: In case the FrTp frame does not require the whole length its PDU (Fr N-PDU) have (e. g. a First Frame or possibly the last Consecutive Frame in a transfer), the remaining space (bits) in the PDU shall be set to 0.

FRTP019: 1 Byte Addressing Not compliant to ISO

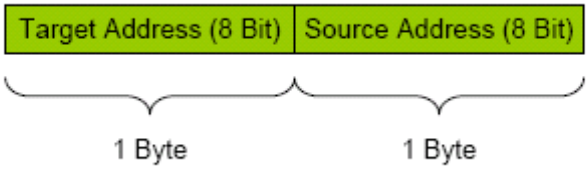


Figure 13: Address header for 1 Byte addressing

For both target and source address 1 Byte is provided, so up to 256 receivers are addressable.

FRTP020: 2 Byte Addressing Not compliant to ISO

Figure 14: Address header for 2 Byte addressing

Looking at this scheme it is possible to address up to 65536 different receivers.

As seen in Figure 12, frames generally consist of the address information, protocol control information and the data. The length and content of the protocol control information (PCI) varies from frame type to frame type.

Before explaining the details of each frame, a short overview is given by the following table (the mentioned bytes and nibbles regard to the PCI):

FRTP021:

ISO	Name	1st Nibble	2nd Nibble	2nd Byte	3rd Byte	4th Byte	5th Byte	Description
YES	SF-I	0x0	FRTP_DL	data	data	data	data	ISO Single Frame
NO	SF-E	0x4	Res (0x0)	FRTP_DL	data	data	data	Extended Single Frame
YES	FF-I	0x1	FRTP_DL		data	data	data	ISO First Frame
NO	FF-E	0x5	Res (0x0)	FRTP_DL				Extended First Frame
YES	CF	0x2	FRTP_SN	data	data	data	data	ISO Consecutive Frame
NO	CF2	0x6	FRTP_SN	data	data	data	data	Consecutive Frame used in Retry Channels
YES / NO	FC	0x3	FRTP_FS	FRTP_BS	FRTP_ STmin	--	--	(ISO) Flow Control Frame
NO	AF	0x7	FRTP_FS	FRTP_BS	FRTP_ STmin	FRTP_ ACK (4 Bit) / FRTP_ SN (4 Bit)	--	Acknowledgement Frame

Table 1: Overview of the different frames format

Note: Unused bytes in table 1 shall be set to 0x00.

Endianess
In case a protocol value transmitted over the bus consists of more than 1 Byte (e. g. Source Address and Target Address when using 2-Byte addressing), the endianess shall be Most Significant Byte first, Least Significant Byte last.

7.3.2 Single Frames (SF-x)

FRTP022: A SF is sent when a message does not exceed the available amount of payload of this frame type or if ISO compliance is required. To be compliant with ISO on the one hand and to allow using the possibilities of FlexRay on the other hand, there are two types of Single Frames. In ISO compliant channels only SF-I is allowed, in non ISO compliant channels (i. e. FRTP_LM = FRTP_L4G) only SF-E is allowed.

7.3.2.1 ISO Single Frame (SF-I)

FRTP023: A SF-I looks as follows (address information header is not depicted):

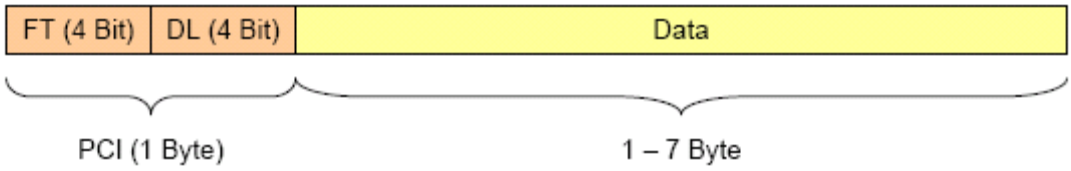


Figure 15: Single Frame ISO

In a SF-I the PCI consists of only one byte. This byte is divided in two parts, called FT (Frame Type) and DL (Data Length). Both parts are 4 Bit long.

The FT field is common to every frame type because it identifies the respective type.

FRTP024: For Single Frames the FT field shall be set to 0x0.

FRTP025: The DL field states the amount of the actual data bytes, according to ISO 15765-2 the values 0x1 – 0x7 (0x6 in ISO6 mode) are valid, so in an ISO compliant connection the size of the associated Fr N-PDU has to be, depending on the addressing mode, 10 (9) or 12 (11) Bytes long, since the SF has this length.

FRTP026: The actual frame length can be derived considering the addressing mode and looking in the length statement of the corresponding FrTp_PduInfoType struct in the configuration.

FRTP027: Including address information the length of an SF-I reaches from 4 Byte (1 Byte pay-load, 1 Byte addressing) to 12 Bytes.

FRTP028: Error Handling

DL field:
Incoming SF-I frames with an invalid DL value of 0x0 or higher than 0x7 (0x6 in ISO6) shall be ignored. This shall also be done if a value arrives which is higher than the amount of payload that can be derived from the length statement of the corresponding FrTp_PduInfoType struct in the configuration and the addressing mode.
If acknowledgement is configured, additionally an AF with a negative acknowledgement shall be sent back to the sender in the cases above.

7.3.2.2 Extended Single Frame (SF-E)

This subchapter is **Not compliant to ISO**.

FRTP029: An SF-E allows using the whole possible FlexRay payload of 254 Bytes for an un-segmented transfer. It looks as depicted in Figure 16:

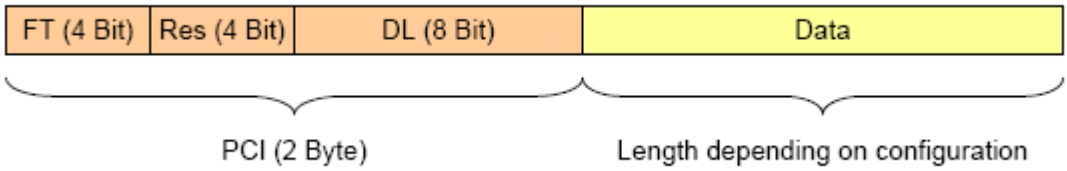


Figure 16: Single Frame Extended

FRTP030: The PCI of an SF-E consists of two bytes. The FT field is 4 Bit long, for an SF-E it shall be set to 0x4. The following nibble is reserved, it shall be set to 0x0..

FRTP031: The next byte is the DL field and states the amount of payload contained in the SF-E. Depending on the configuration of the addressing mode (1 Byte or 2 Byte) and the length of the associated Fr N-PDU, all values except 0x00 and above 0xFA (1 Byte addressing) or above 0xF8 (2 Byte addressing) are valid here.

FRTP032: The minimum length of such a frame is 5 Byte (1 Byte addressing, 1 Byte payload), the maximum is 254 Byte (FlexRay limit according to [12]). The actual frame length can be derived considering the addressing mode and looking in the length statement of the corresponding PDU-Info Struct.

FRTP033: Error Handling

DL field:
If this field contains the value 0x00 or, depending on the addressing mode, a value higher than 0xFA or higher than 0xF8, the SF-E shall be ignored.
If acknowledgement is configured, additionally an AF with a negative acknowledgement shall be sent back to the sender.

General:
If messages longer than allowed by ISO are not configured (FRTP_LM) for the corresponding channel, this frame shall be ignored. This shall also be done if a value arrives which is higher than the amount of payload that can be derived from the length statement of the corresponding PDUInfo Struct and the addressing mode or if a value different from 0x0 arrives in the reserved nibble.
If acknowledgement is configured, additionally an AF with a negative acknowledgement shall be sent back to the sender in the cases above.

7.3.3 First Frames (FF-x)

If a message does not fit into a SF it has to be segmented.

FRTP034: The FlexRay Transport Layer takes the decision whether a message has to be segmented based on the message length, the possibility (depending on per channel configuration) to use SF-E frames and the size of the assigned Fr N-PDU (see also chapter 7.5.1). Therefore to start the transfer of such a long message, a First Frame is used.

FRTP035: To enable compliance with ISO on the one hand and to allow messages longer than 2^{12}-1 Byte on the other hand, there are several types of First Frames.

FRTP036: **Not compliant to ISO**
{
It can be statically per channel configured, whether a First Frame can also start a segmented message in an 1:n connection.
}

7.3.3.1 First Frame ISO (FF-I)

The figure below shows the layout of a FF-I:

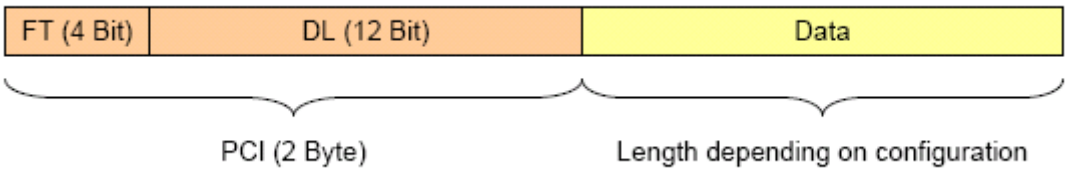


Figure 17: First Frame ISO

In an FF-I the PCI consists of 2 Bytes. As in an SF, the FT field is 4 Bit long, the DL field 12 Bit.

FRTP037: For a FF-I, the FT field shall be set to 0x1.

The DL field contains the length of the whole message. Due to the 12 Bit length of this field, messages up to 2^{12}-1 Bytes can be transferred.

FRTP038: The overall length of a First Frame including address information lasts (depending on the per channel configuration) from 4 Byte to a connection specific maximum.
This maximum on its part depends on the use case (e. g. for communication with CAN for which full ISO compliance is necessary, it will be 10 or 12 (9 or 11 in ISO6)) as well as on the size of the associated Fr N-PDU. The actual amount of payload of an FF-I can be derived by considering the addressing type (1 or 2 Byte) and e. g.

looking in the length designation of the corresponding PDU-Info Struct.

FRTP039: Error Handling

DL field:
Incoming FF-I frames with DL = 0x000 shall be ignored. Moreover if the DL value is lower than the possible (from the PDU size, the addressing type and the channel specific Long Messages switch derivable) payload of a SF, the frame shall also be ignored.
If acknowledgment is configured, in all the cases above additionally an AF with a negative acknowledgement shall be sent back to the sender.

7.3.3.2 First Frame Extended (FF-E)

This subchapter is **Not compliant to ISO**.

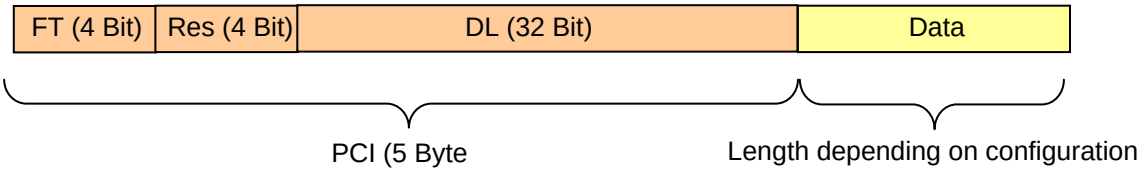


Figure 18: First Frame Extended

In an FF-E the PCI consists of 5 Bytes. The FT field is 4 Bit long, 4 Bits are reserved, the DL field 32 Bit.

FRTP054: The DL field is 4 Byte long, so it allows transporting up to 2^{32}-1 bytes.

FRTP055: The FT field is set to 0x5.

FRTP056: The Res field (reserved) is set to 0x0.

The overall length of an FF-E reaches from 7 Byte to a connection specific maximum which depends on the size of the associated Fr N-PDU.

FRTP057: Error Handling

DL:
If the FR_DL value is lower than the possible (from the PDU size and the addressing type derivable) payload of an SF, the frame shall be ignored.
If acknowledgement is configured for the corresponding channel, an AF with a negative acknowledgement shall be sent back to the sender.

7.3.4 Consecutive Frames

FRTP058: If no error occurred, an FF-x is followed by Consecutive Frames until the whole message is transmitted.

FRTP059: Not compliant to ISO
{
If configured of the specific channel, a Consecutive Frame can also appear in an 1:n connection.
}
As shown below, Consecutive Frames consist of one byte PCI and the payload.

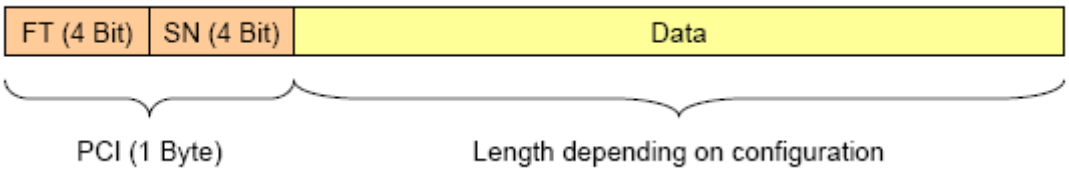


Figure 19: Consecutive Frame

The PCI of a Consecutive Frame consists of one byte which is divided in two 4 Bit parts.

FRTP060: The FT field again states the frame type, for a CF it shall be set to 0x2, for a CF2 it shall be 0x6 (CF2 frames are Not compliant to ISO).

FRTP061: The SN (Sequence Number) field gives the current sequence number of the Consecutive Frame. **Please note that the SN of the CF that immediately follows the FF-x is set to 1** and then incremented with each frame until it wraps around to 0 and so on..

FRTP062: The overall length of a Consecutive Frame including address information ranges (depending on the per connection configuration) from 4 Byte to a connection specific maximum. This maximum on its part depends on the use case (e. g. for communication with CAN for which full ISO compliance is necessary it will be 10 or 12 (9 or 11 in ISO6)) as well as on the size of the associated PDU.
So, the receiving peer can derive the actual data length by looking in the associated PDU-Info Struct und considering the addressing mode.

FRTP063: Error Handling

SN field:
If no acknowledgement is configured, then in case of a wrong SN, i. e. after SN x does not follow SN x+1, the transfer shall be aborted and within *PduR_FrTpRxIndication* the result FRTP_WRONG_SN shall be returned.
If acknowledgment is configured, after the block end a negative acknowledgement shall take place and then the transfer shall be aborted as described above.

If Retry is configured too, then the transfer shall not be aborted but the Retry shall take place (up to FRTP_MAX_RN times).

7.3.5 Flow Control (FC)

FRTP064: A Flow Control frame is used in segmented 1:1 connections (see chapter 7.2.1.1). Thus it cannot appear in a 1:n connection. It allows the receiver to send information to the sender and it allows the sender to send information to the receiver (for Transmit Cancellation). It is sent after reception of an FF-x and after the last CF of a block if no error occurred.

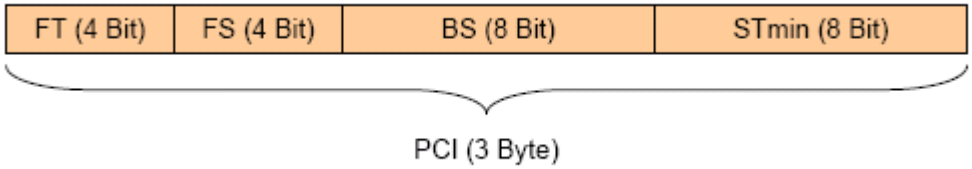


Figure 20: Flow Control frame

FRTP065: A Flow Control frame only consists of Protocol Control Information.

FRTP066: As usual, the FT field states the frame type, thus for Flow control frames it shall be set to 0x3.

FRTP067: In the FS field the parameter FRTP_FS is contained. Depending on per channel configuration three or six different values are possible here (see also chapter 7.2.1.1):

- CTS (value 0x0): Clear To Send
  The sender can continue transmitting the message.
- WT (value 0x1): Wait
  The sender shall wait for another FC frame (and therefore restart its timer FRTP_TIMEOUT_B). If the number of consecutive Flow Control frames with FRTP_FS = WT reaches a per channel defined maximum, the transfer shall be aborted.
- OVFLW (value 0x2): Overflow
  The transfer shall be aborted, because the receiver has not enough buffer for the whole message available (according to the value of the DL field of the FF-x)
- CNLDO (value 0x5): Cancellation Data Outdated (**Not compliant to ISO**)
  The transfer shall be aborted, because the carried

data are outdated. This can be triggered by the upper layer.

- CNLNB (value 0x6): Cancellation No Buffer (**Not compliant to ISO**) The transfer shall be aborted, because no further buffer can be provided to the FlexRay Transport Layer. This can be triggered by the upper layer.
- CNLOR (value 0x7): Cancellation Other Reason (**Not compliant to ISO**) The transfer shall be aborted due to another reason.

FRTP068: BS contains the parameter FRTP_BS, which states the block, size (the number of CFs between the Flow Control frames). If no acknowledgement is configured, all values from 0x00 to 0xFF are valid whereas 0x00 indicates that no more flow control shall take place and the rest of the pending message will be transmitted within one big block. Otherwise, only the values 0x01 – 0x10 are valid.

FRTP069: The last byte contains FRTP_STMIN, which states the minimum gap between two CFs. The valid values are from 0x00 – 0x7F (Separation Time in milliseconds) and from 0xF1 to 0xF9 (separation time of 100 µs, 200 µs, …).

FRTP070: Depending on addressing configuration, a Flow Control frame is 5 or 7 byte long.

FRTP071: Error Handling
FS:
If no Transmit Cancellation for the respective channel is activated, all values higher than 0x2 shall lead to the abortion of the transfer and *PduR_FrTpTxConfirmation* shall be called with the result FRTP_INVALID_FS. Otherwise values higher than 0x7 and the values 0x3 and 0x4 shall lead to the mentioned behaviour.
If acknowledgment with Retry is configured, instead of abortion of the transfer, the frame shall be ignored.

BS:
All values are valid if no acknowledgement is configured. Otherwise only the values from 0x1 to 0x10 are valid. If no Retry is configured in the latter case the transfer shall be aborted and *PduR_FrTpTxConfirmation* shall be called with FRTP_INVALID_BS, otherwise the frame shall be ignored.

STmin:
The invalid values of this parameter range from 0x80 to 0xF0 and from 0xFA to 0xFF. If such a value is received the value 0x7F shall be taken instead.

General:
If the FC is intended for Transmit Cancellation (value 0x5, 0x6 or 0x7 of the FS field), the value of the STmin and BS field shall be 0x0. If at least one of these two fields has another value, the frame shall be ignored.

7.3.6 Acknowledgement Frame (AF)

This subchapter is **Not compliant to ISO**.

FRTP072: If acknowledgement is configured, every block of CFs is in the case of a positive acknowledgement acknowledged by an FC frame (as it is in unacknowledged connections). Additionally an SF-x, the last block of CFs and, in the case of a negative acknowledgement, all other blocks are acknowledged by an AF in 1:1 connections. This frame type cannot appear in an 1:n connection.
This type of frame looks similar to an FC frame (chapter 7.3.5) but it has an additional byte.

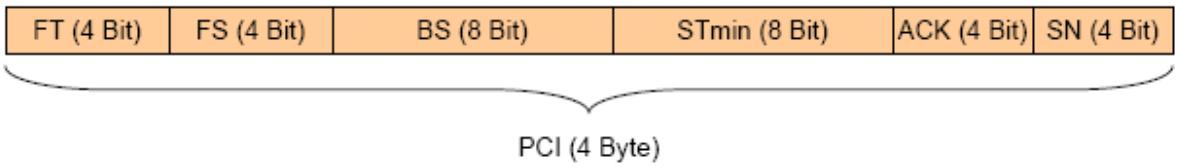


Figure 21: Acknowledgement Frame

FRTP073: This frame is identified by the value 0x7 of the FT field.

FRTP074: The FRTP_FS parameter (FS field) is the same as in an FC frame without the non ISO compliant values.

FRTP075: FRTP_BS (BS field) can only be set to the values 0x01 to 0x10 due to the 4 Bit Sequence Number counter in a CF (chapter 7.3.4).

FRTP076: STmin is the same as in FC frames.

FRTP077:FRTP_ACK (ACK field) gives the type of the acknowledgement, Positive (0x0) or Negative (0x1). All other values are reserved.

FRTP078: FRTP_SN (SN field) contains the number of the first faulty CF within the last block. All values are valid.

FRTP079: Depending on addressing type this frame is 6 or 8 Byte long.

FRTP080: Error Handling:
The following only holds if an AF arrives when it is expected. Otherwise, see chapter 7.3.7.

FS field:
In a segmented transfer, all values higher than 0x2 shall lead to the abortion of the transfer and *PduR_FrTpTxConfirmation* shall be called with the result FRTP_INVALID_FS.
If additionally Retry is configured, such values shall not lead to the abortion of the transfer but to ignore the frame.

BS field: The value 0x00 and all values higher than 0x10 shall cause the abortion of the transfer and *PduR_FrTpTxConfirmation* shall be called with the result FRTP_INVALID_BS.
If additionally Retry is configured, such values shall not lead to the abortion of the transfer but to ignore the frame.

STmin field: The invalid values of this parameter range from 0x80 to 0xF0 and from 0xFA to 0xFF. If such a value is received, the value 0x7F shall be taken instead.

ACK field: Values higher than 0x1 are invalid and shall cause the abortion of the transfer and *PduR_FrTpTxConfirmation* shall be called with the result FRTP_INVALID_ACK.
If additionally Retry is configured, such values shall not lead to the abortion of the transfer but to ignore the frame.

SN field: If here a value arrives which contains a SN of a CF which has not been transmitted within the block, e. g. block size is 10 and this field has value 12, the transfer shall be aborted and *PduR_FrTpTxConfirmation* shall be called with the result FRTP_WRONG_SN.
If additionally Retry is configured, the transfer shall not be aborted but the frame shall be ignored.

General: If for the channel no acknowledgement is configured, this frame type shall be ignored.
In an unsegmented acknowledged transfer, the expected value for the fields BS, STmin and SN is 0x0. Other values shall be tolerated.

For the FS field there is an exception: In case an AF with negative acknowledgement and FS = OVFLW arrives in an **unsegmented** acknowledged transfer or at the end of an segmented acknowledged transfer at the sender, regardless of Retry being configured or not, the transfer shall be aborted and *PduR_FrTpTxConfirmation* shall be called with the result FRTP_OVFLW.

For a better understanding, the following table depicts the possible combinations (and their meaning) of the FS and ACK field in an Acknowledgment Frame:

Possible combinations of FS and ACK field in Acknowledgement Frames	ACK = 0x0	Meaning / Appearence	Leads to Retry (if configured)	ACK = 0x1	Meaning / Appearence	Leads to Retry (if configured)
FS = CTS	X	Positive Acknowledge after SF or after end of Segmented Transfer	NO	X	Negative Acknowledge after SF or after block end in Segmented Transfer	YES
FS = WT	--	--	NO	X	Negative Acknowledge after SF (if currently no Receive buffer is available)	NO
FS = OVFLW	--	--	NO	X	Negative Acknowledge after SF (if no Receive buffer is available)	NO

Table 5: Possible combinations of FS and ACK field

7.3.7 Error Handling of the FT Field

FRTP081: Not every frame type is accepted at any point in time and in any configuration of a channel/connection. Thus a detailed description is given below.

FRTP082: A value of the FR_FT field higher than 0x7 shall always be ignored.

FRTP083: If the corresponding channel and connection is set to be ISO compliant, then the following table holds:

TP Layer Status	***SF-I***	***FF-I (1:1)***	***CF (1:1)***	***FC***	***Other frames***
Segmented Transmit within this channel in progress	If reception is in progress within the channel, see corresponding cell below. Otherwise process the SF-I as start of a new reception.	If reception is in progress within the channel, see corresponding cell below. Otherwise process the FF-I as start of a new reception.	If reception is in progress within the channel, see corresponding cell below. Otherwise ignore it.	If awaited then process, otherwise ignore it.	Ignore
Segmented Receive within this channel in progress	Terminate the current reception, report a *PduR_FrTpRxIndication* with the result FRTP_UNEXP_FRAME and process the SF-I as the start of a new reception.	Terminate the current reception, report a *PduR_FrTpRxIndication* with the result FRTP_UNEXP_FRAME and process the FF-I as the start of a new reception.	If awaited then process, otherwise ignore	If transmission is in progress within the channel, see corresponding cell above. Otherwise ignore it	Ignore
Idle	Process the SF-I as the start of a new reception	Process the FF-I as the start of a new reception	Ignore	Ignore	Ignore

Table 2: FT Error Handling in ISO compliant channels/connections

FRTP084: Otherwise, the behaviour is explained below:

SF-x, FF-x, CF/CF2 and FC: The behaviour shall be as depicted in Table 2 (also for 1:n connections) if no Transmit Cancellation is allowed.
Otherwise if not in the idle state, incoming FCs shall be inspected if it is an FC intended for Transmit Cancellation (not in an unsegmented transfer). The mechanism of the Transmit Cancellation is explained in chapter 7.5.5. If it is not intended for Transmit Cancellation, then the behaviour shall be as depicted in Table 2.

The ignoring of an FF-E shall be according to the value of FRTP_LM.

Regarding CF and CF2 frames there is a special error handling in case Retry is configured (otherwise CF2 frames are ignored):
If the sender starts a block with another frame than expected, i. e. CF instead of CF2 or CF2 instead of CF, then the sender is doing a Retry which has not been requested by the receiver (maybe because of loosing the FC-CTS frame on the bus). So the receiver always has to remember the old block size and send another FC-CTS at the end of this retransmitted block. Errors in the unnecessarily retransmitted block shall be ignored.

AF: If no acknowledgement is activated this frame shall be ignored. Otherwise on the receiver side or in idle state, these frames shall be ignored, too. On the sender side, the behaviour in case of an incoming AF shall be the following:

- If an AF arrives when it is expected, the action is as described in chapter 7.2.1.3 and in section error handling of chapter 7.3.6.
- If a non-faulty AF with positive acknowledgement arrives during a block, it shall be ignored.
- If a non-faulty AF with negative acknowledgement arrives during a block, it shall be processed depending on the activation of the Retry mechanism. If no Retry is configured the transfer shall be aborted. Otherwise the AF shall be processed, i. e. starting with the stated sequence number the Retry shall take place.
- If a faulty AF arrives during a block, it shall be ignored.

7.3.8 Addressing Errors

FRTP085: SF-x:
No restrictions.

FRTP086: FF-x and CF:
If not explicitly configured (by the parameter FRTP_GRPSEG) for the particular channel, a FF-x or CF in a 1:n connection shall be ignored.

FRTP087: FC and AF:
These frame types are not allowed to appear in 1:n connections, thus they shall be ignored in that case.

7.4 Channels and Connections

Within the FlexRay Transport Layer a two-level abstraction concept for communication exists: channels and connections.

7.4.1 Channel

A channel is a group of connections sharing several properties, e. g. acknowledgement, Retry, long messages etc. (see chapter 10.4).

FRTP191: The FlexRay Transport Layer supports several channels. These channels can work concurrently, thus each of them requires its own state machine and management data structures and its own PDU-IDs. The array FRTP_PDU defines the PDUs for sending and receiving data. The PDU FRTP_PDU_FC defines the PDU which can be used for transmitting Flow Control or Acknowledgement Frames (especially with large TP PDUs this is an advantage, because the Flow Control or Acknowledgement Frames need only to be as large as necessary and not e. g. 150 Bytes too as the other TP frames (PDUs)). The TP also accepts incoming Flow Controls / Acknowledges on an ID out of FRTP_PDU, so it is not prescribed to use this special PDU (because this generates more effort in scheduling the TP PDU). This is determined by the switch FRTP_USE_PDU_FC.
Furthermore each channel can have different properties so for each channel its own configuration data is needed.

FRTP088: The FlexRay Transport Layer shall be implemented to support multiple channels. This means that an implementation shall provide at least up to 32 channels being able to work concurrently. The exact number shall be configurable by a compile switch, named FRTP_CHAN_NUM.

7.4.2 Connection

A connection within a channel identifies the sender and the receiver(s) of this particular communication. A connection can belong to only one channel and inherits all the properties of its channel. Additionally there are some properties which are configurable for each connection independently e.g. the ID of the to be used FrTp PDUs of the corresponding channel (see chapter 10.4). Within a channel the different connections do not work concurrently (with the exception of being able to provide Full Duplex capacity), only connections located in different channels can work concurrently, since there is a dedicated state machine for each channel.

FRTP184: The FlexRay Transport Layer shall be implemented to support a post compile time per channel configurable number of connections. The exact number shall be configurable by the channel parameter FRTP_CON_NUM. Additionally depending on FRTP_USE_PDU_FC for sending / receiving Flow Control or Acknowledgement Frames the PDU FRTP_PDU_FC is used.

FRTP089: Two or more connections with both the same sender address AND the same receiver address are ONLY supported if these two connections are located in pairwise different channels. The N-SDU Ids (FRTP_SDUID) have to be unique over all channels!

The following picture illustrates the relationship between connections and channels:

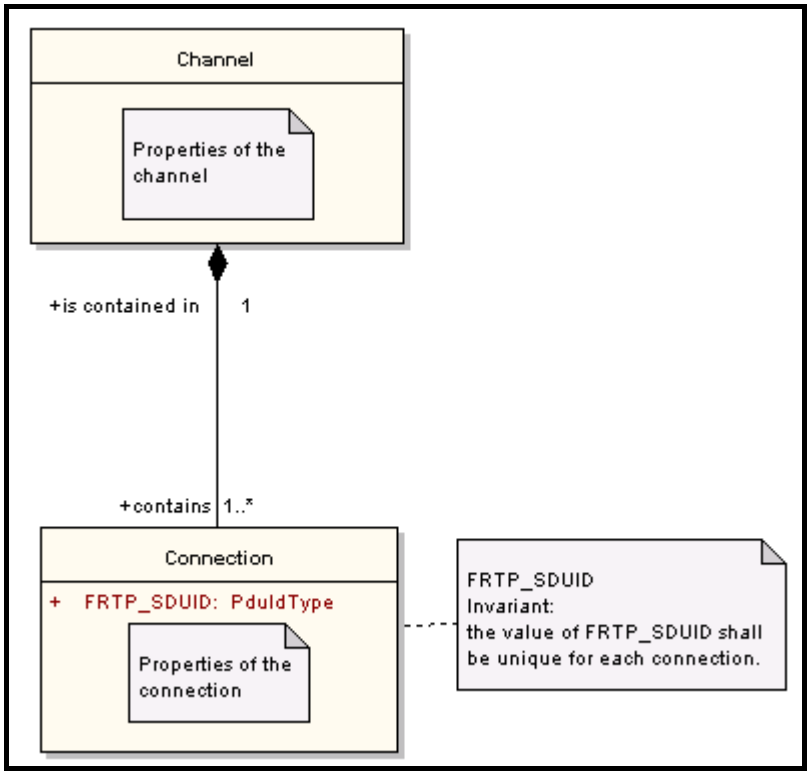


Figure 22: Relationship between channels and connections

7.4.3 Required Buffer within a Channel

Since the Transport Layer has to concatenate its header information and the payload, there is some amount of buffer needed.
The size of this buffer is n, where $n = 2 * \sum(Length\ of\ PDU_i)$, where i iterates over all PDUs configured for the respective channel (the multiplier 2 arises for Full Duplex channels).

Because of connections within a channel cannot work concurrently, this buffer can be used both for sending and for copying received PDUs to the FlexRay Transport Layer.

7.4.4 Identifying a Channel at Reception of an N-PDU

As mentioned above, each channel has its own PDU-IDs, thus the PDU-ID shall be utilized to identify the corresponding channel. This is necessary in order to determine whether the contained TP frame uses 2 or 4 byte addressing.

7.4.5 Full Duplex and Half Duplex

FRTP192: The FlexRay Transport Layer is intended to provide Full Duplex capacity in each channel (Full Duplex within a channel means: If a connection is transmitting messages, it shall be possible that within another or the same connection a reception is possible, too). So in Full Duplex channels, the sending “part” and the receiving “part” of the statemachine shall be able to work concurrently, even within the same connection.
Because of the Full Duplex capacity of a channel, a situation in which the currently sending “part” of the local channel sends an FF-x or last CF of a block (which has to be sent in the PDU having the TxConfirmation configured, see FRTP174: and FRTP188:) and the currently receiving “part” of the local channel transmits an FC or AF (which has also to be sent in the PDU having the TxConfirmation configured) could arise.
In this situation both “parts” of the channel have to be synchronized, e. g. the receiving part shall not send the FC (or AF) until the transmitting part has got the TxConfirmation for the PDU.

In case the implementation shall only support Half Duplex Channels, Table 2 has to be slightly modified. These modifications can be found in the corresponding table in [10].

FRTP193: However, please be aware of the following issues concerning Half Duplex Channels: Imagine a Transmission has been initiated and an FF-x already been sent. Now the local peer is waiting for an FC frame.
Unfortunately, within the same channel a remote peer has also started a segmented Transmission and thus also sent an FF-x nearly simultaneously (NOTE: Simultaneously means with respect to the mapping of the TP PDUs into FlexRay cells, so “simultaneously” can mean “with a time-lag up to a few milliseconds”).
According to [10], the (at both peers) incoming FF-x has to be ignored, thus a timeout of Timer BS will occur on both sides.
If additionally Retry is configured for the channel, both sides will try again after a configurable amount of time. In order not to produce this “FF-x Crash” again, the timeouts on both sides have to be different with the respect to the scheduling of the PDUs of each side. Thus it can be necessary to configure a difference of several milliseconds between both wait times. In larger networks quite a big gap between the timeout times might arise and complex consistence checks between the individual wait times could be necessary.

7.5 Further Principles of Working

7.5.1 Decision of Segmentation

FRTP090: As mentioned earlier in this specification, there are several factors influencing the decision of the FrTp to segment a message (Fr N-SDU) or not.
The values of the following parameters play a role hereby:
FRTP_PDU_LENGTH, FRTP_LM, FRTP_ADRTYPE, FRTP_MULT_REC, FRTP_GRPSEG and the length of the to be transmitted message (Fr N-SDU).

FRTP091:FRTP_PDU_LENGTH states the length of a TP frame (Fr N-PDU) on the physical layer. The amount of bytes of FRTP_PDU_LENGTH which is usable for payload, i. e. for the Fr N-SDU, depends on the length of the PCI of the used frames, i. e. if two or four bytes (FRTP_ADRTYPE) are needed to state to address information. The frames which are allowed to be utilized and the payload they can carry depend on the value of FRTP_LM (e.g. SF-E is allowed or not, SF-I can carry 7 or 6 bytes etc.). In case the connection is an 1:n connection (FRTP_MULT_REC) the parameter FRTP_GRPSEG states whether segmentation is allowed or not.
With all these information and the length of the to be transmitted Fr N-SDU the FrTp can decide whether it has to segment the Fr N-SDU or not.

7.5.2 Multiple Fr N-PDUs for one connection, mapping of Fr N-PDU to a connection

FRTP092: If more than one Fr N-PDU is utilized for the Fr N-SDU within a connection, the FlexRay Transport Layer shall use them in ascending order.
This is necessary to avoid CFs coming out of order in a segmented transfer (see also chapter 10.4.6).

FRTP188: In order to allow high bandwidth and being able to prevent out-of-order arriving of FrTp frames (which are not necessarily FlexRay frames!) a Tx Confirmation from the FlexRay Interface shall be given only each time when having sent the last Fr N-PDU of the group of PDUs the respective connection uses (see also FRTP182:). This together with FRTP174: allows reaching the above mentioned goals.
Please note:
Only PDUs of the same size shall be used within a connection, the FrTp does not deal with CFs of different size in a connection.
Therefore it is necessary to configure a TxConfirmation in the FlexRay Interface for exactly one PDU (the one with the highest ID) of each size used in the respective channel (see also FRTP182:).

FRTP189: In the case of using more than one Fr N-PDU for a connection, the Timer AS (chapter 10.4.1) starts to run after calling *FrIf_Transmit()* for the first PDU of the group.

FRTP190: In the case of using more than one Fr N-PDU for a connection and sending a message (or the remainder of a message) which is not long enough for needing all Fr N-PDUs of the connection, the necessary number of PDUs, starting

with the smallest ID, shall be skipped (e. g. if PDUs 4, 5, 6, 7 belong to a connection and currently only 2 PDUs are needed, then PDU 6 and 7 shall be used for sending). This is necessary since the TxConfirmation configured for the last PDU of the group (see FRTP188:) is needed to stop Timer AS. Of course this holds also for sending FC or ACK frames (i. e. for these the PDU having the TxConfirmation configured shall be used). The AS Timer has to be started on the first used PDU.

FRTP093: The mapping of the arrived frame (Fr N-PDU) is done by comparing the Target Address (see FRTP019: , FRTP020:) of the received frame with the Local Address (see FRTP168: FRTP_LA) of the local connections (of the by the means of the PDU-Id identified channel) and comparing the Source Address (see FRTP019: , FRTP020:) of the received frame with the Remote Address (see FRTP168: FRTP_RA) of the local connections (of the by the means of the PDU-ID identified channel). If both fit within a connection, the frame shall be processed for this connection, otherwise the frame will be ignored..

7.5.3 Sending and Receiving within the same connection (Fr N-SDU Id)

FRTP094: The FlexRay Transport Layer shall be implemented to support both sending and receiving within one connection at one peer (do not mistake this with full duplex). So the same connection can be utilized for sending and receiving.
To explain it more in detail, imagine a connection being in idle state. If now the call *FrTp_Transmit()* occurs, the local peer becomes the sender in this connection (Source Address of TP frame = FRTP_LA, Target Address of TP frame = FRTP_RA). Otherwise, if an *FrTp_RxIndication()* for an Fr N-PDU Id which is mapped on the Fr N-SDU Id of this connection occurred, it would become the receiver (Source Address of TP frame = FRTP_RA, Target Address of TP frame = FRTP_LA).

This feature is intended for connections in which sometimes one peer has to send data and sometimes the other in order not to need two connections in this case.

7.5.4 Behavior on Timeouts and Errors when calling the FlexRay Interface

FRTP095: The behavior in case a timeout occurs depends on the value of FRTP_ACKTYPE, i.e. what kind of acknowledgement is configured for the corresponding channel.

Please note that the behavior in case of a FrIf error (return value E_NOT_OK of *FrIf_Transmit*) shall be in all three following cases the same as if the AS / AR timer expires (of course the result in *PduR_TxConfirmation* / *PduR_RxIndication* shall be FRTP_FRIF_ERROR). For Start and Stop of the different timers see chapter 10.4.1.

7.5.4.1 No Acknowledgement configured for the Channel

FRTP096: In this case, the behavior shall be as described in [10], i.e.:

- If the AS timer (FRTP_TIMEOUT_AS) expires, depending on the value of FRTP_MAX_AS, sending shall be retried (because of still remaining attempts) or the transmission shall be aborted and within *PduR_TxConfirmation* the result FRTP_TIMEOUT_A shall be returned.
- If the AR timer (FRTP_TIMEOUT_AR) expires, depending on the value of FRTP_MAX_AR, sending shall be retried (because of still remaining attempts) or the transmission shall be aborted and within *PduR_RxIndication* the result FRTP_TIMEOUT_A shall be returned.
- If the BS timer (FRTP_TIMEOUT_BS) expires, the transmission shall be aborted and within *PduR_TxConfirmation* the result FRTP_TIMEOUT_B shall be returned.
- If the CR timer (FRTP_TIMEOUT_CR) expires, the transmission shall be aborted and within *PduR_RxIndication* the result FRTP_TIMEOUT_C shall be returned. If previously in the current block a sequence error ocurrred, at the blockend this error will be reported in *PduR_RxIndication*.

7.5.4.2 Acknowledgement without Retry configured for the Channel

This subchapter is **Not compliant to ISO**.

FRTP097: In this case, the behavior is the following:

- In case of a timeout of timer AS, AR or BS, the behavior shall be as mentioned in chapter 7.5.4.1.
- If the CR timer (FRTP_TIMEOUT_CR) expires, an AF with negative acknowledgement shall be sent, the transmission shall be aborted and within *PduR_RxIndication* the result FRTP_TIMEOUT_C shall be returned. If previously in the current block a sequence error ocurrred, at the blockend this error will be reported in *PduR_RxIndication*.

7.5.4.3 Acknowledgement with Retry configured for the Channel

This subchapter is **Not compliant to ISO**.

FRTP098: In this case, the behavior shall be the following:

- In case of a timeout of timer AS or AR, the behavior shall be as mentioned in chapter 7.5.4.1.
- If the BS timer (FRTP_TIMEOUT_BS) expires, the sender shall retransmit the whole block up to FRTP_MAX_RN times. After that, the transmission shall be aborted and within *PduR_TxConfirmation* the result FRTP_TIMEOUT_B shall be returned.
- If the CR timer (FRTP_TIMEOUT_CR) expires, the receiver shall send an AF with negative acknowledgement and the sequence number of the missed CF.

This shall be done up to FRTP_MAX_RN times. After that, the transmission shall be aborted and within *PduR_RxIndication* the result FRTP_TIMEOUT_C shall be returned. If previously in the current block a sequence error ocurrred, at the block end this error will be reported in *PduR_RxIndication*.

7.5.5 Transmit Cancellation

This subchapter is **Not compliant to ISO**.
FRTP099: This feature can be (de)activated by static configuration (parameter FRTP_TC). Transmit Cancellation is triggered by the call of *FrTp_CancelTransmitRequest*.
This call shall set the TC_REQUEST flag of the corresponding channel).
In order to allow a fast cancellation, this flag has to be checked twice at the sender side:
Every time before trying to get a new Tx buffer and every time before calling *FrIf_Transmit*.
At the receiver side, the flag shall also be checked twice:
Every time before trying to get a new Rx buffer and before calling *PduR_RxIndication*. At this side, the triggering of the cancellation is not possible in an 1:n connection, since the receiver is not allowed to send a frame in such a connection.

FRTP100: The service works at the sender side of a connection as follows:

- If no transmit request is pending for the corresponding connection, there is nothing to do.
- If a request is pending but the transmission has not been started, the corresponding TC_REQUEST flag (see chapter 7.5.8.2) will be set. Thus the transfer won't take place.
- If the transmission already has been started, the corresponding TC_REQUEST flag (see chapter 7.5.8.2) will be set. Thus a FC for Transmit Cancellation (see chapter 7.3.5) is sent and the transfer will be aborted.

Note that the last option is only possible in a segmented transfer, because in an unsegmented transfer there is only one frame to be sent and after the call of *FrIf_Transmit* there is no possibility to stop the sending.

FRTP101: At the receiver side of a connection, this is possible only for 1:1 connections, since only in this connection type the receiver is also allowed to send frames. Here the service works as follows:

- If no reception is ongoing, there is nothing to do.
- Otherwise the corresponding TC_REQUEST flag (see chapter 7.5.8.2) will be set. Thus no Rx Indication will be done. If it is a segmented transfer and a 1:1 connection, an FC for Transmit Cancellation (see chapter 7.3.5) is sent. The transfer will be aborted.

FRTP102: If an FC frame for Transmit Cancellation arrives (sender or receiver), the transfer shall be aborted and *PduR_FrTpTxConfirmation* / *PduR_FrTpRxIndication* shall be called with the appropriate result, i. e. FRTP_CNLDO, FRTP_CNLNB or FRTP_CNLOR.

FRTP103: Please note, that if a transfer was cancelled by the call of *FrTp_CancelTransmitRequest*, there will be no additional call of *PduR_FrTpTxConfirmation* / *PduR_FrTpRxIndication* at the side (Sender / Receiver) at which the cancellation was initiated.

7.5.6 Parameter Changing

FRTP104: The FlexRay Transport Layer also supports the in [10] mentioned optional service for changing the parameter `FRTP_STMIN` by the API call *FrTp_ChangeParameterRequest*. A change is not possible during an ongoing reception and will lead to the result value FRTP_RX_ON in *PduR_FrTpChangeParameterConfirmation*.
Please note that against [10] only the value of STmin is changeable. This comes from the buffer requesting concept of AUTOSAR which requires an automatic choosing of the block size by the FrTp software module.

7.5.7 Buffer Requests, Block Size and WAIT-Frames

As mentioned earlier, the FlexRay Transport Layer does not provide message buffers, neither for sending nor for receiving. Instead it requests either a transmit buffer or a receive buffer in order to fulfill the request.

FRTP221: In case a reception is being started, there is a difference between the first request for a receive buffer and the subsequent requests.
Within the first request, the TP states the minimum length it expects for that buffer (this is needed to avoid a buffer smaller than the payload of an CF from which the block size is derived). This minimum length is valid for all subsequent requests until the ongoing transfer has been finished. In the mentioned subsequent requests, the TP states the number of valid data bytes the currently used receive buffer.
Further information can be found in the following subchapters.

FRTP105: Depending on the message length and configuration of the FrTp a segmented or an unsegmented transfer will take place.

7.5.7.1 Unsegmented Transfer

FRTP106: At the sender side, this principle works as follows:

1. At the FrTp the service *FrTp_Transmit* is called.
2. Therefore the FrTp will call *PduR_FrTpProvideTxBuffer* (value 0 for Length what means buffer can be of arbitrary size) in order to get data bytes to send.

FRTP107: If not all data bytes are contained within the first Tx buffer, the service *PduR_FrTpProvideTxBuffer* will be called again and again (always with value 0 for the Length parameter) until all data bytes are put into the SF. If calling the service does not provide a valid buffer, it is tried up to FRTP_MAX_BUFREQ times to get one (if the call returned BUFREQ_E_BUSY) or the transfer shall be aborted (*PduR_FrTpTxConfirmation* shall be called with FRTP_NO_BUF).

FRTP108: At the receiver side, the principle works as follows:

1. The FrTp gets an *FrTp_RxIndication* by the FrIf
2. Therefore the FrTp will call *PduR_FrTpProvideRxBuffer* (with value 0 for the length in the **PduInfoPtr which means that the provided buffer can be of arbitrary size) in order to get a buffer for the to be received data.

The **PduInfoPtr has to be provided by the TP.

FRTP109: If *PduR_FrTpProvideRxBuffer* provides a valid buffer, it will be filled with data and, if the buffer is too small, another one will be requested. If calling the service does not provide a valid buffer, it is tried up to FRTP_MAX_BUFREQ times to get one (if the call returned BUFREQ_E_BUSY) or the transfer shall be aborted (return value was BUFREQ_E_OVFL or BUFREQ_E_NOT_OK) and *PduR_FrTpRxIndication* shall be called with FRTP_NO_BUF.
If acknowledgement is configured in case of failing to get a receive buffer (either BUFREQ_E_OVFL or BUFREQ_E_NOT_OK was returned or after FRTP_MAX_BUFREQ attempts no buffer is available), an AF with a negative acknowledgement and FS = OVFLW is sent back to the sender.
During the mentioned attempts to get a buffer, an AF with negative acknowledgement and FS = WT is sent to the sender (in order to prevent a timeout of the BS timer). Up to FRTP_MAX_BUFREQ AF frames with FS = WT can be sent before the transfer is aborted.

7.5.7.2 Segmented Transfer

FRTP110: At the sender side, this principle works as follows:

1. At the FrTp the service *FrTp_Transmit* is called.
2. Therefore, the FrTp will call *PduR_FrTpProvideTxBuffer* in order to get data bytes to send.

FRTP111: When calling the latter service, the value of the API parameter Length is important.
If no Retry is configured, the value 0 shall be used, since it does not matter what size the transmit buffer has because no data have to be kept in store for a retry.
If Retry is configured, things behave differently. Before requesting a transmit buffer, the sender sends an FF-x (without data bytes) to the receiver and waits for an FC frame to get knowledge about the block size (and the STmin value). The sender then can request a transmit buffer with an appropriate value for Length, i. e. the amount of data bytes which will be transferred in this block. This is necessary to have all the

data available in case a Retry takes place. Since the sender will get a buffer with the requested size, there will be no additional buffering in the FrTp necessary.
Since the requested transmit buffer can theoretically go up to 4016 bytes (in case the receiver has as much receive buffer and the Fr N-PDUs are configured to maximum FlexRay payload length of 254 bytes) the parameter FRTP_MAXBS allows limiting the the block size to a defined maximum per channel.

FRTP112: After transmitting the block, the sender waits for the next FC and can then request a buffer according to the new block size. The FrTp will request Tx buffers as long as there are data bytes to send.

FRTP113: If calling the service does not provide a valid buffer, it is tried up to FRTP_MAX_BUFREQ times to get one (if the call returned BUFREQ_E_BUSY) or the transfer shall be aborted if the call returns BUFREQ_E_NOT_OK or BUFREQ_E_OVLW (*PduR_FrTpTxConfirmation* shall be called with FRTP_NO_BUF).

FRTP114: At the receiver side, this principle works as follows:

1. The FrTp gets an *FrTp_RxIndication* by the FrIf
2. Therefore the FrTp will call *PduR_FrTpProvideRxBuffer* in order to get a buffer for the to be received data (with value “payload size of one CF” for the length in the **PduInfoPtr) in order to get a buffer for the to be received data.

The **PduInfoPtr has to be provided by the TP for the first call (since it states the minimum length in **PduInfoPtr).

FRTP115: Depending on the size of the buffer returned by the latter API call, the FrTp uses a block size which makes the transferred data bytes within the upcoming block to fit into the provided receive buffer. Of course, the provided buffer can have a size which is between x * Payload_in_CF and (x+1) * Payload_in_CF (where X is the number of CFs sent in the block. This does not matter since the FrTp has to set the value SduLength within the PduInfo Struct of the provided Rx buffer which states the number of valid bytes in the buffer. The block size, which is derived from the length of the provided buffer, is then sent to the sender within an FC frame.
FRTP116: If calling the service does not provide a valid buffer, it is tried up to FRTP_MAX_BUFREQ times (always after FRTP_TIME_BUFFER) to get one if the call returned BUFREQ_E_BUSY (and a WAIT frame (i. e. FC frame with FS = WT) is sent). Otherwise (if something different than BUFREQ_E_BUSY is returned) the transfer shall be aborted (*PduR_FrTpRxIndication* shall be called with FRTP_NO_BUF).

FRTP117: In case of failing to get a receive buffer, an FC with FS = OVFLW is sent back to the sender.

7.5.7.3 Buffer Locking

FRTP118: At the sender side the provider of the transmit buffer (e.g. DCM or AUTOSAR COM) shall not access the buffer, until the next one has been requested

(by *PduR_FrTpProvideTxBuffer*) or transmission has been completed (i.e. calling of *PduR_FrTpTxConfirmation*).

FRTP119: At the receiver side the provider of the receive buffer (e.g. DCM or AUTOSAR COM) shall not access a buffer, until the next one has been requested (by *PduR_FrTpProvideRxBuffer*) or reception has been completed (i.e. calling of *PduR_FrTpRxIndication*).

7.5.7.4 Data Bytes in First Frames

FRTP120: Not compliant to ISO {
As implicitly quoted in 7.5.7.2, if acknowledgement with Retry is configured for the corresponding channel, no payload is sent within an FF-x if a segmented transfer takes place. This is necessary because the FF-x is sent before the request for a Tx buffer (because the FrTp needs the block size from the receiving peer in order to call *PduR_FrTpProvideTxBuffer* with the correct value for Length).
}

FRTP121: If acknowledgment without Retry (or no acknowledgement) is configured, there are data bytes within an FF-x.

7.5.8 Counters, Flags and Actions

7.5.8.1 Counters

FRTP122: There are several counters to count the different Retry attempts. Each counter has its individual maximum value in order give much flexibility here. Each of these counters is reset every time when the corresponding retry is successful.

FRTP123: counter_RN
This counter counts the sending retries initiated due to a frame error, e. g. bad SN in a CF.
It is limited by the value of FRTP_MAX_RN.

FRTP124: counter_BUFREQ
This counter counts the local buffer requesting retries initiated due to not getting an Tx or Rx buffer (depending on the return value of the corresponding buffer request function, see chapter 7.5.7).
It is limited by the value of FRTP_MAX_BUFREQ.

FRTP125: counter_WT
This counter counts the remote buffer requesting retries, i. e. the arriving WAIT frames.
It is limited by the value of FRTP_MAX_BUFREQ.

FRTP126: counter_FRIF
This counter counts the attempts to send a Fr N-PDU via *FrIf_Transmit()* in case this call return E_NOT_OK.

It is limited by the value of FRTP_MAX_FRIF.

FRTP127: counter_AR
This counter counts the attempts to send a Fr N-PDU (FC, AF), by resetting Timer AR, in case a timeout of Timer AR occurs.
It is limited by the value of FRTP_MAX_AR.

FRTP128: counter_AS
This counter counts the attempts to send a Fr N-PDU (SF-x, FF-x, CF, FC (in case of Transmit Cancellation)), by resetting Timer AS, in case a timeout of Timer AS occurs.
It is limited by the value of FRTP_MAX_AS.

FRTP129: counter_BS
This counter counts, depending on if retry is configured or not, the retries to send the last block (or the SF if acknowledgement is configured) again in case a timeout of the timer BS occurs. Another point of view is that this counter counts the timeouts of the timer BS.
It is limited by the value of FRTP_MAX_RN.

FRTP130: counter_CR
This counter counts, depending on if retry is configured or not, the retries to send an AF in case a timeout of the Timer CR occurs.
It is limited by the value of FRTP_MAX_RN.

7.5.8.2 Flags

There will be several flags within this software module (see sequence diagrams in chapter 9). In order not to go too deep into implementation details, they are described only briefly in the following.

FRTP131: TX_PDU_AVAILABLE flag
This flag exists for every connection. It is set by the call *FrTp_Transmit* and indicates the availability of the Fr N-SDU for the corresponding connection. Thus it can be considered when processing the Tx request. It is cleared before the call of *PduR_FrTpTxConfirmation*.

FRTP132: RX_PDU_AVAILABLE flag
This flag exists for every Fr N-PDU which is configured to be received by the FlexRay Transport Layer, so there can be more than one such flag in a connection. It is set by the call *FrTp_RxIndication* and indicates the availability of the Fr N-PDU for the corresponding connection. Thus it can be considered when processing the Rx indication. In an unsegmented transfer, it is cleared before the call of *PduR_FrTpRxIndication*. In segmented one it is cleared when finished processing the Fr N-PDU.

FRTP133: TC_REQUEST flag
This flag exists for every channel (twice for Full Duplex channels). It is set by the call of *FrTp_CancelTransmitRequest*() (if the service returns E_OK) and processed

before asking for a new buffer or before sending (and calling *PduR_RxIndication* respectively) a frame. The flag is cleared when processing the cancellation request.

FRTP134: ERROR flag
This flag exists for every Fr N-PDU within a channel, so there can be more than one such flag in a connection. It is set by the call *FrTp_RxIndication* when an error in the received frame is detected. The reaction on these errors will be as described in sections “Error Handling” throughout chapter 7.3.
When receiving an Fr N-PDU without an error, this flag shall be cleared.

FRTP135: ERROR_OCCURRED flag
This flag exists for every channel. It indicates that an error occurred during a segmented reception in order to react appropriate at the block end. It is cleared after the reaction (Retry, Negative Acknowledgement, Abortion) at the block end.

7.5.8.3 Actions

There are three main actions within this software module to be done.

FRTP136: Sending
Sending is initiated by the call of *FrTp_Transmit*. Thus all sending related mechanisms described in this document are executed. It is finished when no more TX_PDU_AVAILABLE flag is set.

FRTP137: Receiving
Receiving is initiated by the call of *FrTp_RxIndication*. Thus all receiving related mechanisms described in this document are executed. It is finished when no more RX_PDU_AVAILABLE flag is set, no timeouts have occurred and no ERROR flag is set, i. e. when every ongoing reception either has been completed or aborted.

FRTP138: Timeout supervision
Timeout supervision has always to be done within the sending and receiving related mechanisms. The reaction on timeouts can be found in chapter 7.5.4. When no more timeout to supervise is left, this action can be stopped.

7.5.9 Ignored Frames

FRTP139: Throughout this specification many times the ignoring of frames is mentioned. Please note that an ignored frame does never affect a timer, i.e. never causes the restarting of a timer.

FRTP140: The only exception is at the receiver side when retry is configured and due to an erroneous frame an AF with negative acknowledgement is sent and therefore it is waited for the retry frame(s). In this case, the timer CR will be reset by the erroneous frame.

7.6 Buffer Access Modes in the FlexRay Interface

FRTP187: The FlexRay Transport Layer software module shall be implemented being able to work both with PDUs configured (in the FlexRay Interface) for Immediate Buffer Access and for Decoupled Buffer Access, i. e. it shall reuse its channel specific temporary buffers, in case the local peer is the sender, not before the TxConfirmation for the respective PDU group has been arrived.
In the receiving case, from the FlexRay Transport Layers point of view there is no difference between an Fr N-PDU being configured for Decoupled Buffer Access or Immediate Buffer Access.

7.7 Error classification

This chapter lists and classifies all the errors that can be detected within this software module.

FRTP179: Error classification table

Type or error	*Relevance*	*Related error code*	*Value*
API service called before initializing the module	Development	FRTP_E_NOT_INIT	0x1
API service called with NULL pointer	Development	FRTP_E_NULL_PTR	0x2
API service called with not allowed parameter value	Development	FRTP_WRONG_PARAM_VAL	0x3

7.8 Error detection

FRTP217: The detection of development errors is configurable (*ON* / *OFF*) at pre-compile time.
The switch *FRTP_DEV_ERROR_DETECT* (see chapter 10) shall activate or deactivate the detection of all development errors.

FRTP205: If the *FRTP_DEV_ERROR_DETECT* switch is enabled API parameter checking is enabled. The detailed description of the detected errors can be found in chapter 7.7 and chapter 8.

FRTP218: The detection of production code errors cannot be switched off.

7.9 Error notification

FRTP206: Detected development errors shall be reported to the *Det_ReportError* service of the Development Error Tracer (DET) if the pre-processor switch *FRTP_DEV_ERROR_DETECT* is set (see chapter 10).

8 API specification

8.1 Imported types

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

FRTP141: The following types are defined within AUTOSAR and used for the FlexRay Transport Layer:

Header file	*Imported Type*
BufReq_Types.h	BufReq_ReturnType
ComStack_Types.h	PduIdType
	PduLengthType
	NotifResultType
PrimitiveTypes.h	PduInfoType
Std_Types.h	Std_VersionInfoType
	Std_ReturnType

8.2 Type definitions

FRTP223: The following FrTp specific types shall be defined in FrTp_Types.h

8.2.1 FrTp_CancelReasonType

FRTP143: FrTp_CancelReasonType

Name:	`FrTp_CancelReasonType`	
Type:	`Enumeration`	
Range:	`FRTP_CNLDO`	Cancel Transfer because data are outdated
	`FRTP_CNLNB`	Cancel Transfer because no further buffer can be provided
	`FRTP_CNLOR`	Cancel Transfer because of another reason
Description:	The reason is sent to the other peer (not on receiver side in a 1:n connection) by the means of an appropriate FC frame.	

8.2.2 FrTp_ParameterValueType

FRTP145: FrTp_ParameterValueType

Name:	`FrTp_ParameterValueType`	
Type:	`uint8`	
Range:	`0x00 - 0x7F;` `0xF1 - 0xF9`	FRTP_STmin
Description:	Ranges of this parameter.	

8.2.3 FrTp_ChangeResultType

FRTP146: FrTp_ChangeReslutType

Name:	FrTp_ChangeResultType	
Type:	Enumeration	
Range:	FRTP_OK	Successful execution of the parameter change request
	FRTP_RX_ON	Parameter change request not executed due to an ongoing reception
	FRTP_WRONG_PARAMETER	Parameter change request not executed due to a wrong value for FrTp_ParameterType
	FRTP_WRONG_VALUE	Parameter change request not executed due to a wrong value for FrTp_ParameterValueType
Description:	Values according to ISO 15765-2	

8.2.4 FrTp_CancelResultType

FRTP194:

Name:	FrTp_CancelResultType	
Type:	Enumeration	
Range:	FRTP_OK	Successful execution of the cancel transmit request
	FRTP_E_NOT_OK	Cancellation was not successful, e. g. the FC(CNLxx) could not be sent
Description:	--	

8.2.5 FrTp_PduInfoType

FRTP216:

Name:	FrTp_PduInfoType		
Type:	Structure		
Element:	PduIdType	FrTp_PduId	The PDU-ID
	PduLengthType	FrTp_Pdu_Length	Length of this PDU / < 255
Description:	--		

8.3 Function definitions

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

FRTP207: Here is the API Naming convention for the FrTp services:

- The service name format is FrTp_<ServiceName>(…)
- <ServiceName>: is the name of the service primitive with first letter of each word upper case and consecutive letters lower case

8.3.1 Standard functions

8.3.1.1 FrTp_GetVersionInfo

FRTP215:

Service name:	FrTp_GetVersionInfo
Syntax:	`void FrTp_GetVersionInfo( Std_VersionInfoType* versioninfo )`
Service ID[hex]:	0x27
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	versioninfo Pointer to where to store the version information of this module.
Return value:	None
Description:	Returns the version information.

FRTP202: This service returns the version information of this module. The version information includes:

- Module Id
- Vendor Id
- Vendor specific version numbers (BSW00407).

This function shall be pre compile time configurable `On/Off` by the configuration parameter: `FRTP_VERSION_INFO_API`

Configuration: `FRTP_VERSION_INFO_API`

8.3.2 Initialization and Shutdown

8.3.2.1 FrTp_Init

FRTP147: FrTp_Init

Service name:	FrTp_Init
Syntax:	`void FrTp_Init( )`
Service ID[hex]:	0x00
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This service initializes all global variables of a FlexRay Transport Layer instance and set it in the idle state. It has no return value because software errors in initialisation data shall be detected during configuration time (e.g. by configuration tool). Furthermore, if a hardware error occurs it shall be reported via the error

	manager modules.

Caveats: The call of this service is mandatory before using the FrTp instance for further processing.

8.3.2.2 FrTp_Shutdown

FRTP148: FrTp_Shutdown

Service name:	FrTp_Shutdown

Syntax:	`void FrTp_Shutdown(`
	
	`)`

Service ID[hex]:	0x01
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This service closes all pending transport protocol connections by simply stopping operation, frees all resources and stops the FrTp Module

8.3.3 Normal Operation

8.3.3.1 FrTp_Transmit

FRTP149: FrTp_Transmit

Service name:	FrTp_Transmit	

Syntax:	`Std_ReturnType FrTp_Transmit(`
	`  PduIdType FrTpTxPduId,`
	`  const PduInfoType* PduInfoPtr`
	`)`	

Service ID[hex]:	0x02	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant	
Parameters (in):	FrTpTxPduId	This parameter contains the FlexRay TP instance unique identifier of the Fr N-SDU to be transmitted.
	PduInfoPtr	A pointer to a structure with Fr N-SDU related data: data length and pointer to an Fr N-SDU buffer.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The request has been accepted E_NOT_OK: The request has not been accepted, e. g. due to a still ongoing transmission in the corresponding channel or the to be transmitted message is too long.
Description:	This service is utilized to request the transfer of data. It sets a flag for indicating that a transmit request is present.	

	This function has to be called with the PDU-Id of the FrTp, i.e. the upper layer has to translate its own PDU-Id into the one of the TP for the corresponding message. Within the provided PduInfoPtr only SduLength is valid (no data)! If this function returns E_OK then there will arise an call of PduR_FrTpProvideTxBuffer in order to get data for sending.

8.3.3.2 FrTp_CancelTransmitRequest

FRTP150: FrTp_CancelTransmitRequest

Service name:	FrTp_CancelTransmitRequest	
Syntax:	`Std_ReturnType FrTp_CancelTransmitRequest( PduIdType FrTpTxPduId, FrTp_CancelReasonType FrTpCancelReason )`	
Service ID[hex]:	0x03	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant	
Parameters (in):	FrTpTxPduId	This parameter contains the FlexRay TP instance unique identifier of the Fr N-SDU which transfer has to be cancelled.
	FrTpCancelReason	The reason for cancellation
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Cancellation request of the transfer (sending or receiving) of the specified Fr N-SDU is accepted. E_NOT_OK: Cancellation request of the transfer of the specified Fr N-SDU is rejected, e. g. cancellation is requested at the receiver in an 1:n connection or in an unsegmented transfer at the receiver or cancellation is not allowed for the corresponding channel.
Description:	This service primitive is used to cancel the transfer of pending Fr N-SDUs. The connection is identified by FrTpTxSduId. This function has to be called with the PDU-Id of the FrTp, i. e. the upper layer has to translate its own PDU-Id into the one of the TP for the corresponding message.	

Caveats:
If a cancel request is accepted and cancelling a transfer on the sender side, the function PduR_FrTpTxConfirmation won't be called after finishing (successfully or not) the cancellation.
If a cancel request is accepted and cancelling a transfer on the receiver side, the function PduR_FrTpRxIndication will not be called after finishing (successfully or not) the cancellation.
Instead, if the cancellation request has been accepted, PduR_FrTpCancelTransmitConfirmation will be called when the cancellation is finished (successfully or not).

8.3.3.3 FrTp_ChangeParameterRequest:

FRTP151: FrTp_ChangeParameterRequest

Service name:	FrTp_ChangeParameterRequest	

Syntax:	`void FrTp_ChangeParameterRequest(`
	`  PduIdType FrTpTxPduId,`
	`  FrTp_ParameterValueType FrTpParameterValue`
	`)`	

Service ID[hex]:	0x04	
Sync/Async:	Asynchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	FrTpTxPduId	Gives the ID of the connection (message) for whose channel the change shall be done
	FrTpParameterValue	This parameter contains the new value of FRTP_STMIN
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	This service primitive is used to request the change of the value of the FRTP_STMIN parameter. The new value is given by FrTpParameterValue. This function has to be called with the PDU-Id of the FrTp, i.e. the upper layer has to translate its own PDU-Id into the one of the TP for the corresponding message.	

Caveats: According to ISO 15765-2 this is not possible to change the value of the parameter during an ongoing reception.

8.4 Call-back notifications

8.4.1 FrTp_TriggerTransmit

FRTP154: FrTp_TriggerTransmit

Service name:	FrTp_TriggerTransmit	

Syntax:	`Std_ReturnType FrTp_TriggerTransmit(`
	`  PduIdType FrTxPduId,`
	`  PduInfoType* PduInfoPtr`
	`)`	

Service ID[hex]:	0x07	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	FrTxPduId	ID of FlexRay N-PDU that is requested to be transmitted.
Parameters (inout):	PduInfoPtr	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.
Parameters (out):	None	
Return value:	Std_ReturnType	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.
Description:	This function is called by the FlexRay Interface for sending out a FlexRay frame. The trigger transmit is initiated by the FlexRay schedule.	

	This function has to be called with the PDU-Id of the FrTP, i. e. the FlexRay Interface has to translate its own PDU-Id into the corresponding one of the FrTp.

Caveats: This function might be called in interrupt context

8.4.2 FrTp_RxIndication

FRTP152: FrTp_RxIndication

Service name:	FrTp_RxIndication	

Syntax:	`void FrTp_RxIndication(`
	`  PduIdType FrRxPduId,`
	`  const PduInfoType* PduInfoPtr`
	`)`	

Service ID[hex]:	0x05	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	FrRxPduId	This parameter contains the identifier of the received Fr N-PDU.
	PduInfoPtr	Contains the length (SduLength) of the received I-PDU and a pointer to a buffer (SduDataPtr) containing the I-PDU.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	The FlexRay Interface calls this primitive after the reception of an Fr N-PDU. Within this function, the FlexRay Transport Layer at least copies the received Transport Layer frame to itself. This function has to be called with the PDU-Id of the FrTp, i. e. the FlexRay Interface has to translate its own PDU-Id into the corresponding one of the FrTp.	

8.4.3 FrTp_TxConfirmation

FRTP153: FrTp_TxConfirmation

Service name:	FrTp_TxConfirmation	

Syntax:	`void FrTp_TxConfirmation(`
	`  PduIdType FrTxPduId`
	`)`	

Service ID[hex]:	0x06	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	FrTxPduId	This parameter contains the identifier of the transmitted Fr N-PDU.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	This function is called by the FlexRay Interface after the TP-related Pdu has been	

	transmitted over the network. Within this function, the FlexRay TP shall route this confirmation to the configured target transport connection. All transmitted FlexRay frames belonging to the FlexRay TP shall be confirmed by using this function. This function has to be called with the PDU-Id of the FrTp, i.e. the FlexRay Interface has to translate its own PDU-Id into the corresponding one of the FrTp.

8.5 Scheduled functions

These functions are directly called by Basic Software Scheduler.

8.5.1 FrTp_MainFunction

FRTP162: Main Function

Service name:	FrTp_MainFunction

Syntax:	`void FrTp_MainFunction(`
	
	`)`

Service ID[hex]:	0x10
Timing:	FIXED_CYCLIC
Description:	Schedules the FlexRay TP. (Entry point for scheduling)

FRTP203: The main function for scheduling the TP (Entry point for scheduling)

Terms and definitions:
Fixed cyclic: Fixed cyclic means that one cycle time is defined at configuration and shall not be changed because functionality is requiring that fixed timing (e.g. filters).
Variable cyclic: Variable cyclic means that the cycle times are defined at configuration, but might be mode dependent and therefore vary during runtime.
On pre condition: On pre condition means that no cycle time can be defined. The function will be called when conditions are fulfilled. Alternatively, the function may be called cyclically however the cycle time will be assigned dynamically during runtime by other modules.

8.6 Expected Interfaces

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

8.6.1 Mandatory Interfaces

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

FRTP219:

API function	*Description*
FrIf_Transmit	Requests the sending of a PDU.
PduR_FrTpRxIndication	Rx indicator for the FlexRay TP
PduR_FrTpProvideTxBuffer	ProvidesTx Buffer for the FlexRay TP
PduR_FrTpTxConfirmation	Tx confirmation for the FlexRay TP
PduR_FrTpProvideRxBuffer	Provides Rx Buffer for the FlexRay TP

8.6.2 Optional Interfaces

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

FRTP220:

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

8.6.3 Configurable interfaces

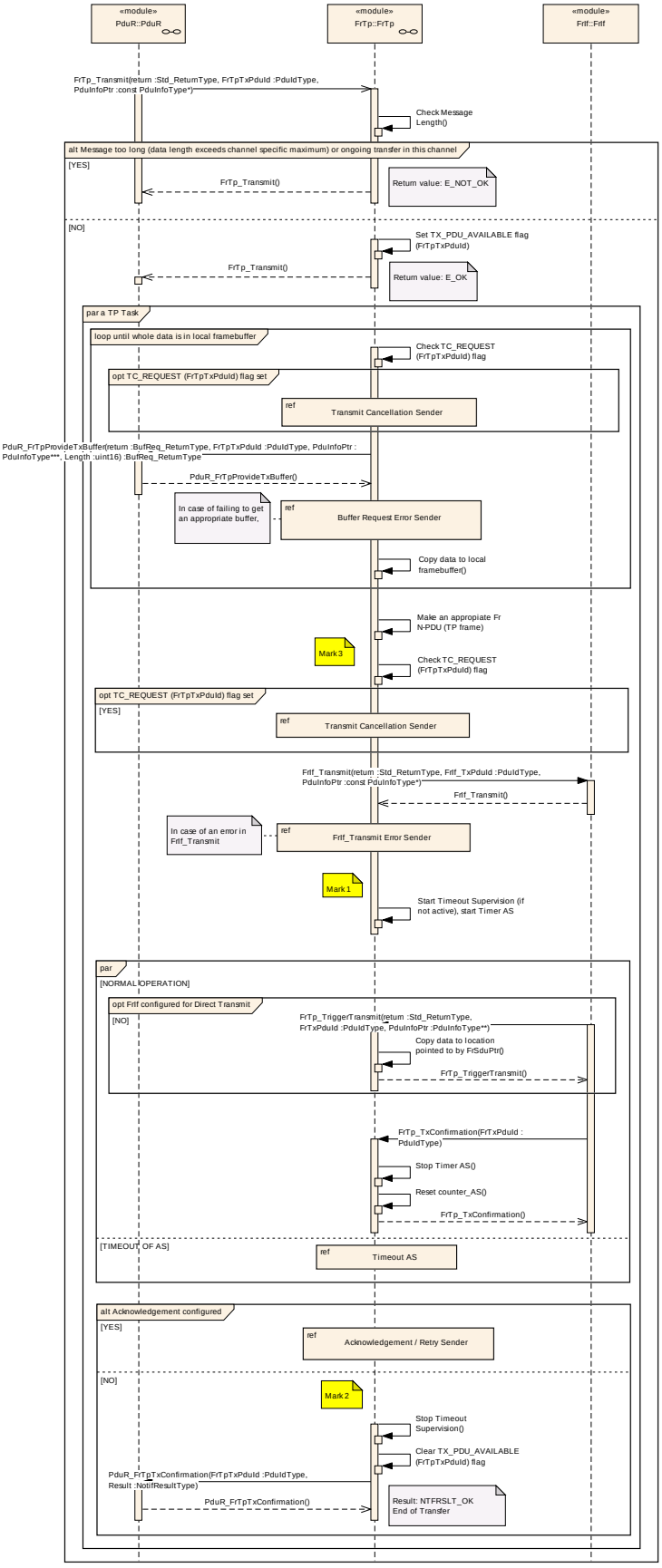
None

9 Sequence diagrams

Although the following sequence diagrams are quite detailed, they do not depict every detail. Thus they should be seen as an addendum to this specification.

9.1 Sending

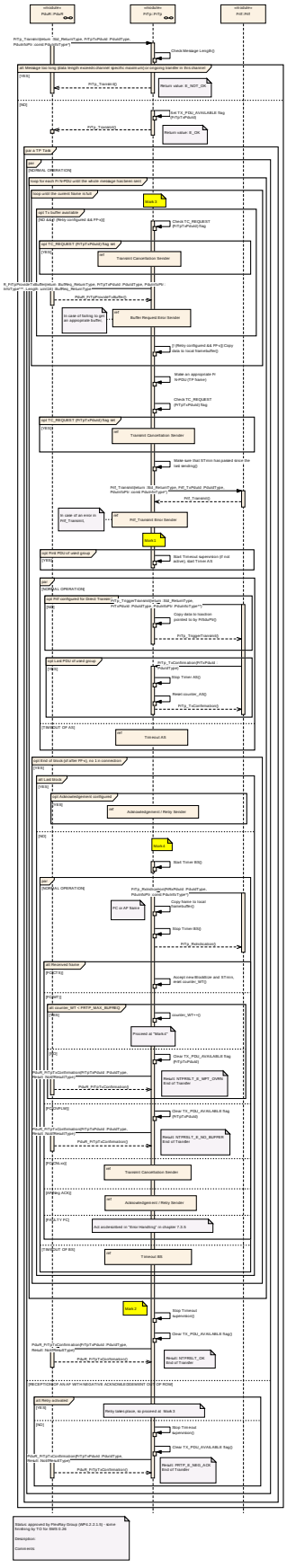
9.1.1 Unsegmented Sending



Status: POSTPONED TO AUTOSAR R2.1 !!!

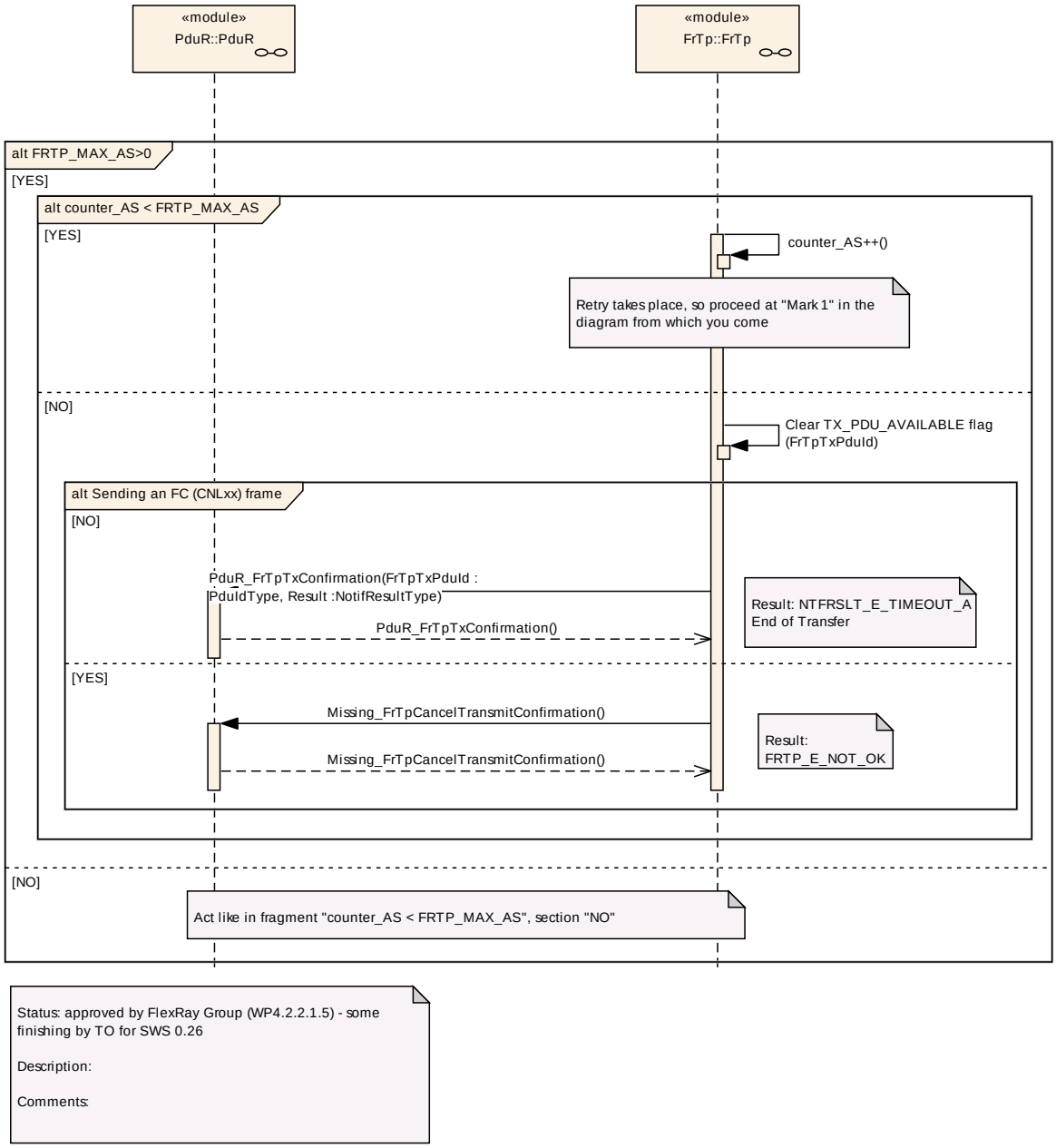
Some finishing by TO for SWS 0.26
Details see "FlexRay UML Sequence Diagrams - Change History.doc"
2006-04-12: Update by BMW_TK to match Frlf SWS V1.2.5
2006-04-13: Update by DECOMSYS_TGAL to match Frlf SWS V1.2.5
2006-04-24: Minor changes and corrections by BMW_TK

9.1.2 Segmented Sending



9.1.3 Others

9.1.3.1 Timeout AS

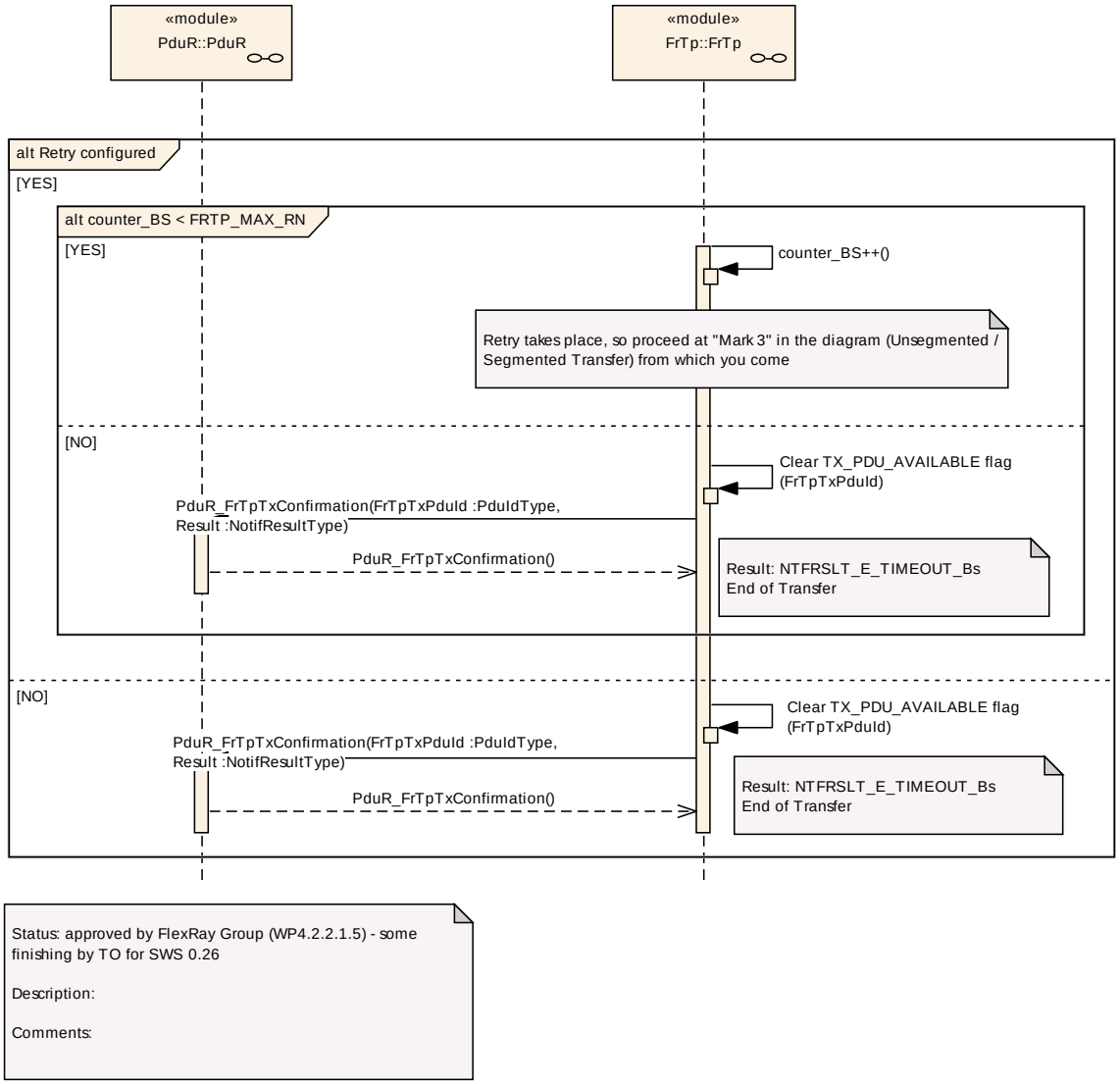


Status: approved by FlexRay Group (WP4.2.2.1.5) - some finishing by TO for SWS 0.26

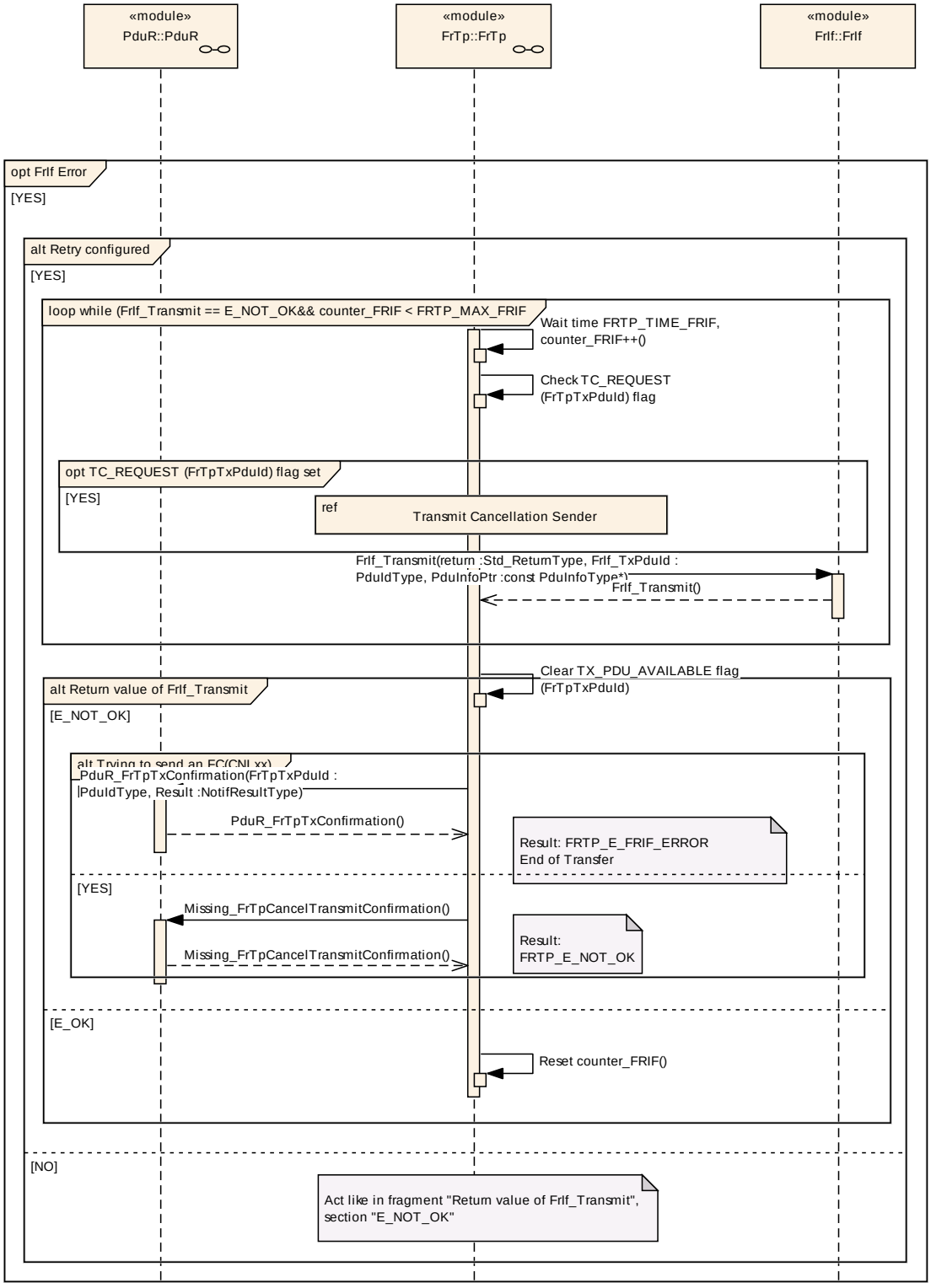
Description:

Comments:

9.1.3.2 Timeout BS



9.1.3.3 FrIf_Transmit Error Sender

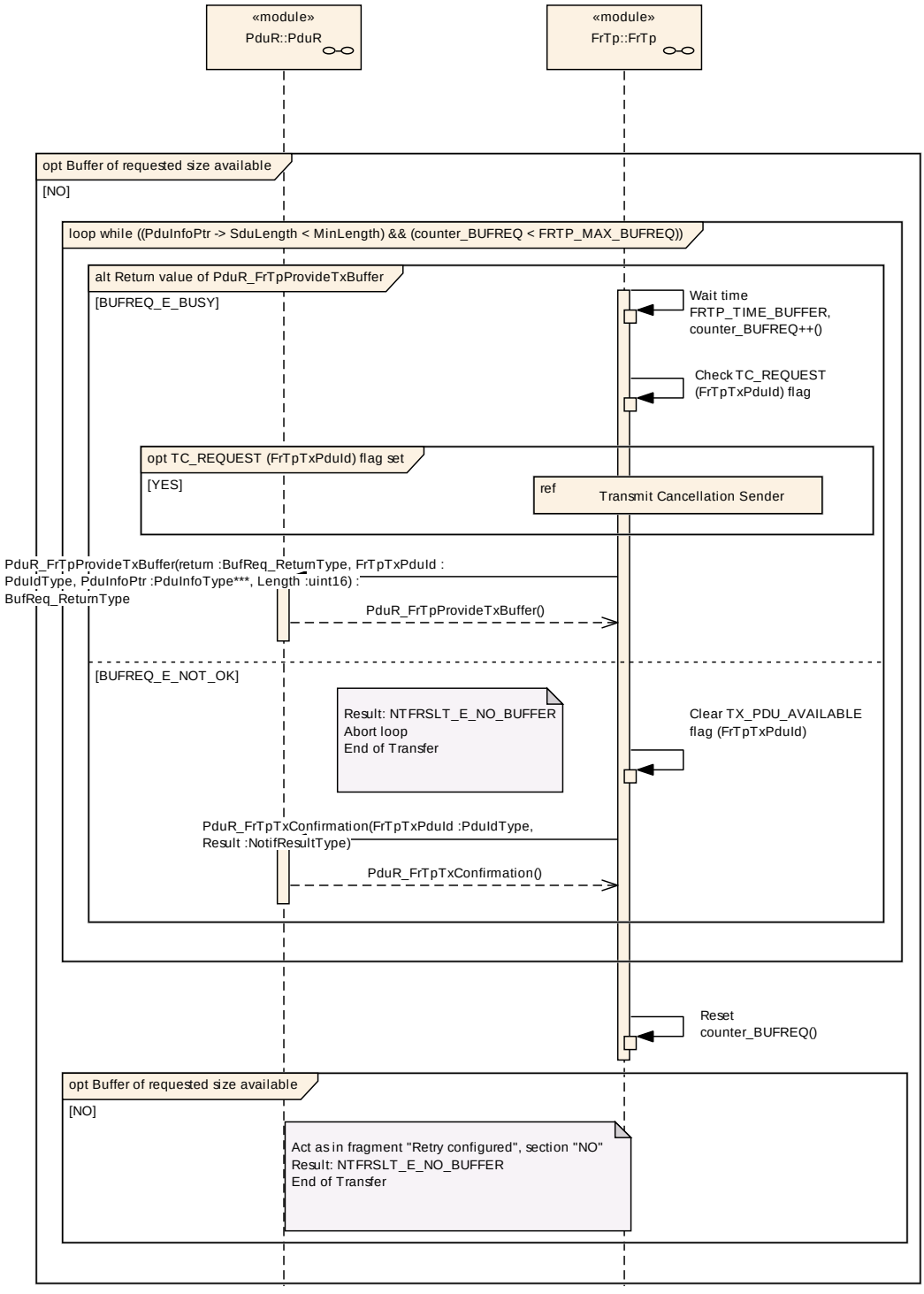


Status: approved by FlexRay Group (WP4.2.2.1.5) - some finishing by TO for SWS 0.26

Description:

Comments:

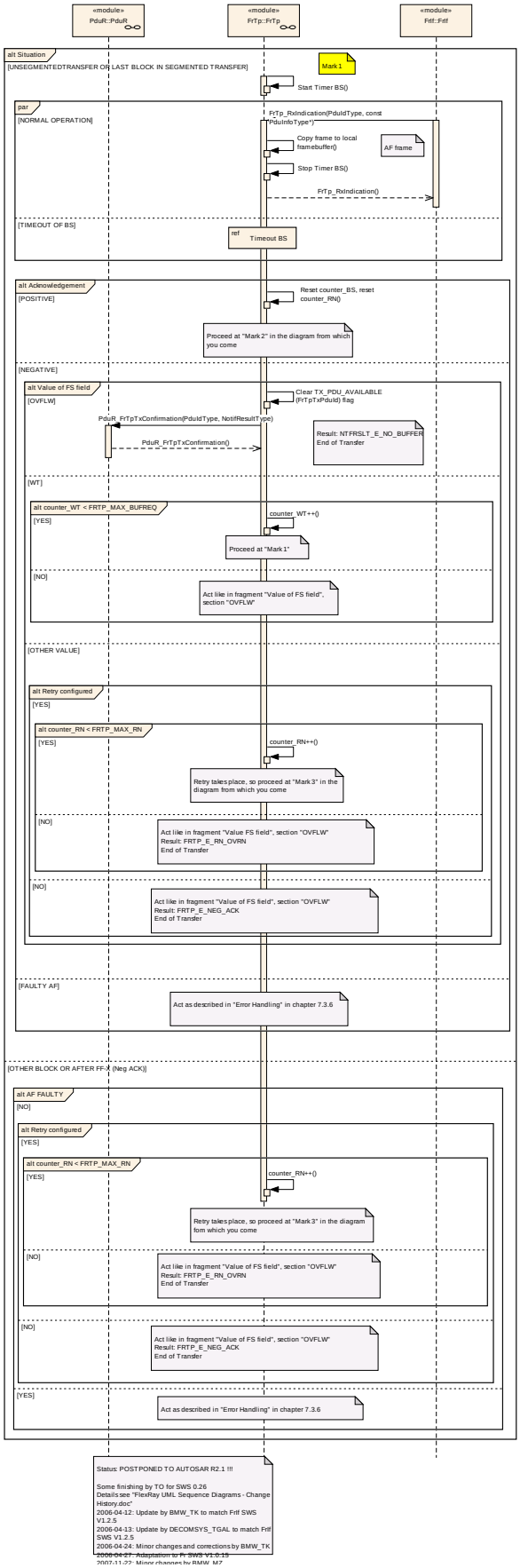
9.1.3.4 Buffer Request Error Sender



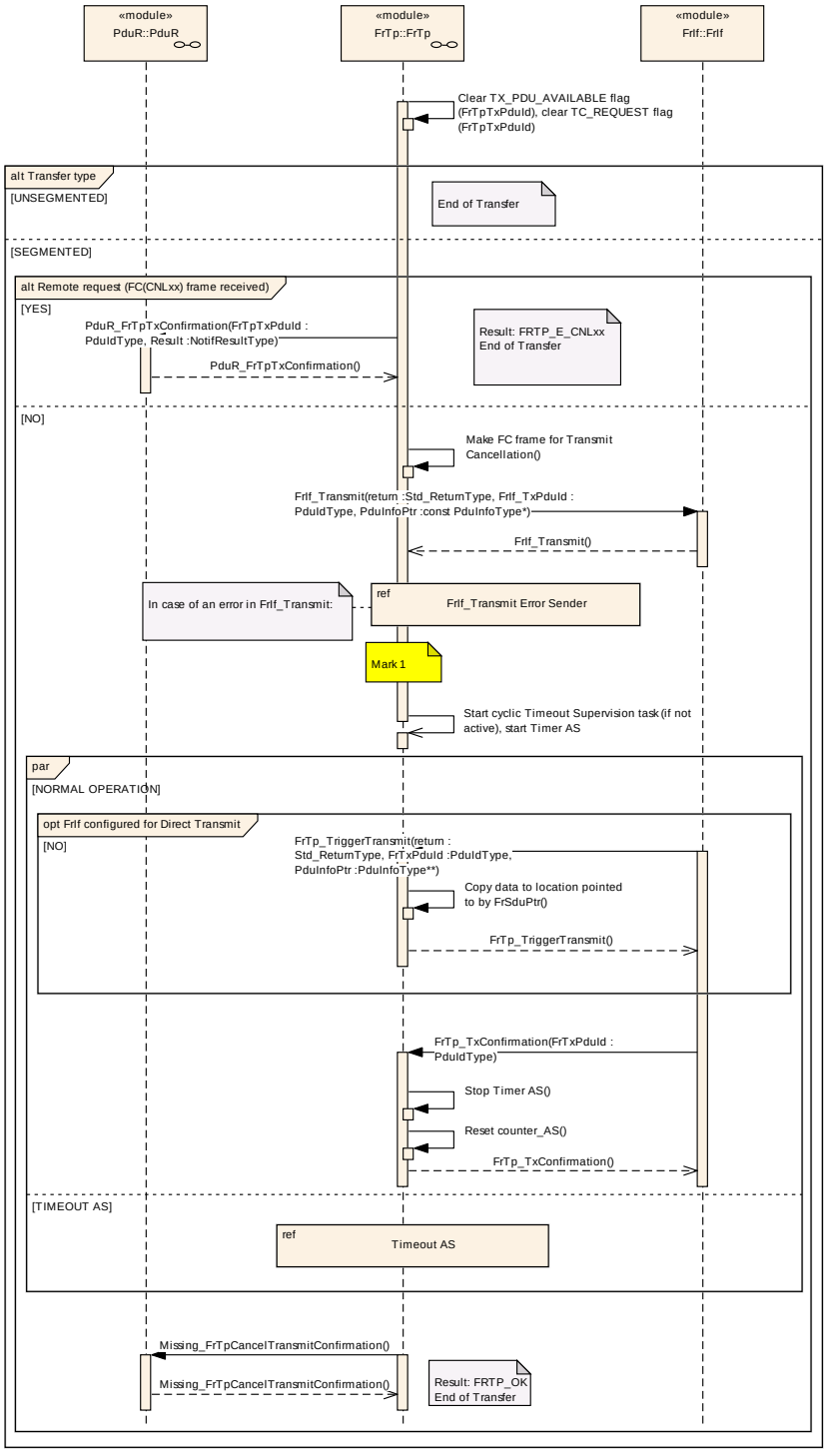
Status: POSTPONED TO AUTOSAR R2.1 !!!

Some finishing by TO for SWS 0.26
Details see "FlexRay UML Sequence Diagrams - Change History.doc"
2006-04-12: Update by BMW_TK to match FrIf SWS V1.2.5
2006-04-13: Update by DECOMSYS_TGAL to match FrIf SWS V1.2.5
2006-04-24: Minor changes and corrections by BMW_TK
2006-04-27: Adaptation to Fr SWS V1.0.15
2007-11-22: Minor changes by BMW_MZ

9.1.3.5 Acknowledgement / Retry Sender



9.1.3.6 Transmit Cancellation Sender



Status: POSTPONED TO AUTOSAR R2.1 !!!

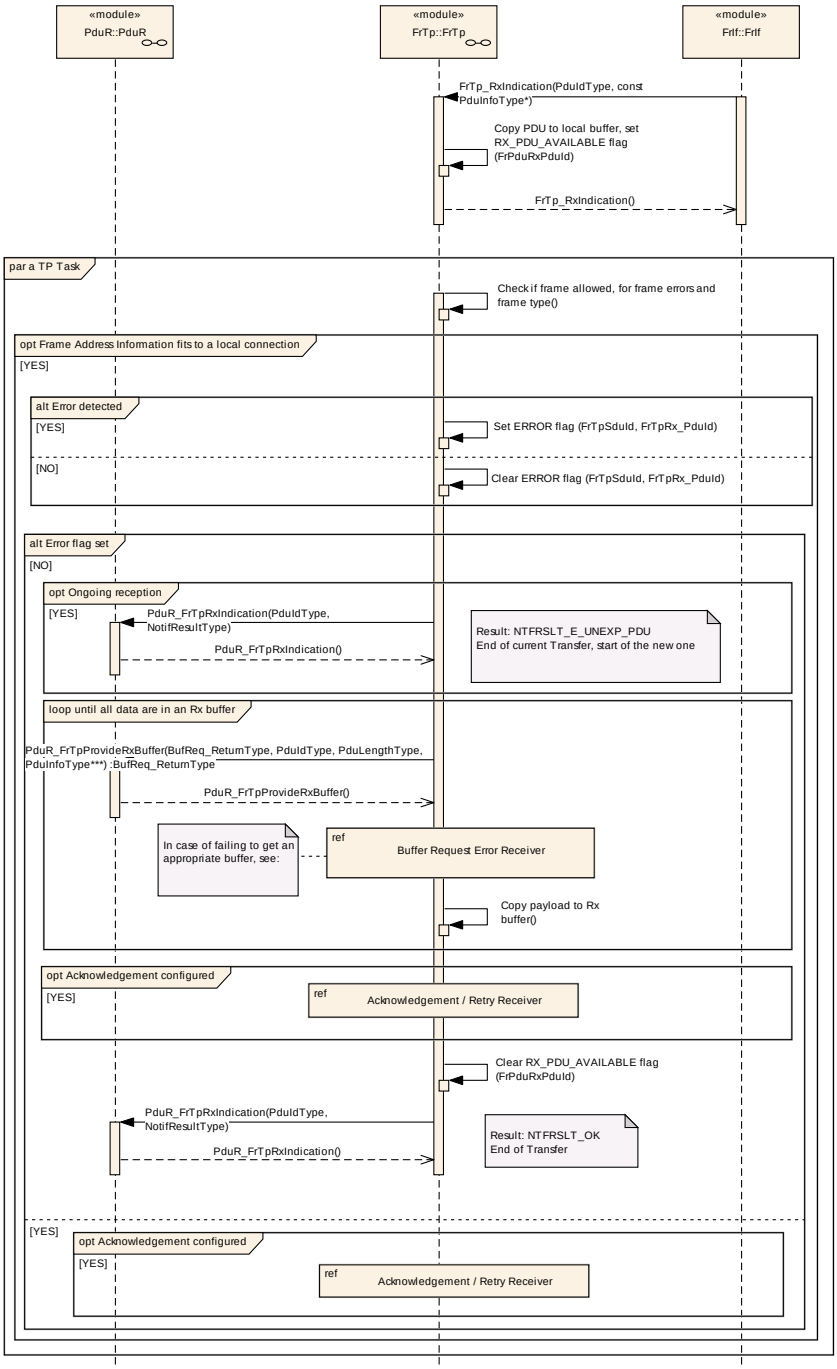
Some finishing by TO for SWS 0.26
Details see "FlexRay UML Sequence Diagrams - Change History.doc"
2006-04-12: Update by BMW_TK to match Frlf SWS V1.2.5
2006-04-13: Update by DECOMSYS_TGAL to match Frlf SWS V1.2.5
2006-04-24: Minor changes and corrections by BMW_TK
2006-04-27: Adaptation to Fr SWS V1.0.15
2007-11-22: Minor changes by BMW_MZ

Description:

Comments:

9.2 Receiving

9.2.1 Unsegmented Receiving



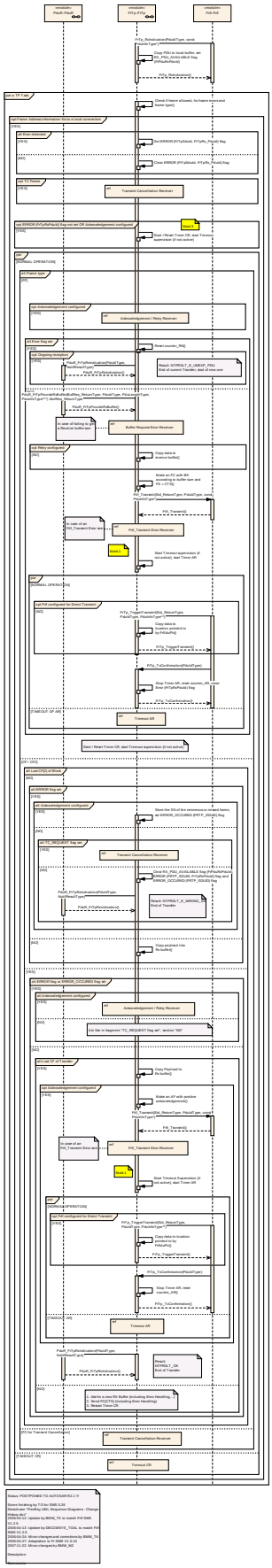
Status: POSTPONED TO AUTOSAR R2.1 !!!

Some finishing by TO for SWS 0.26
Details see "FlexRay UML Sequence Diagrams - Change History.doc"
2006-04-12: Update by BMW_TK to match Frlf SWS V1.2.5
2006-04-13: Update by DECOMSYS_TGAL to match Frlf SWS V1.2.5
2006-04-24: Minor changes and corrections by BMW_TK
2006-04-27: Adaptation to Fr SWS V1.0.15
2007-11-22: Minor changes by BMW_MZ

Description:

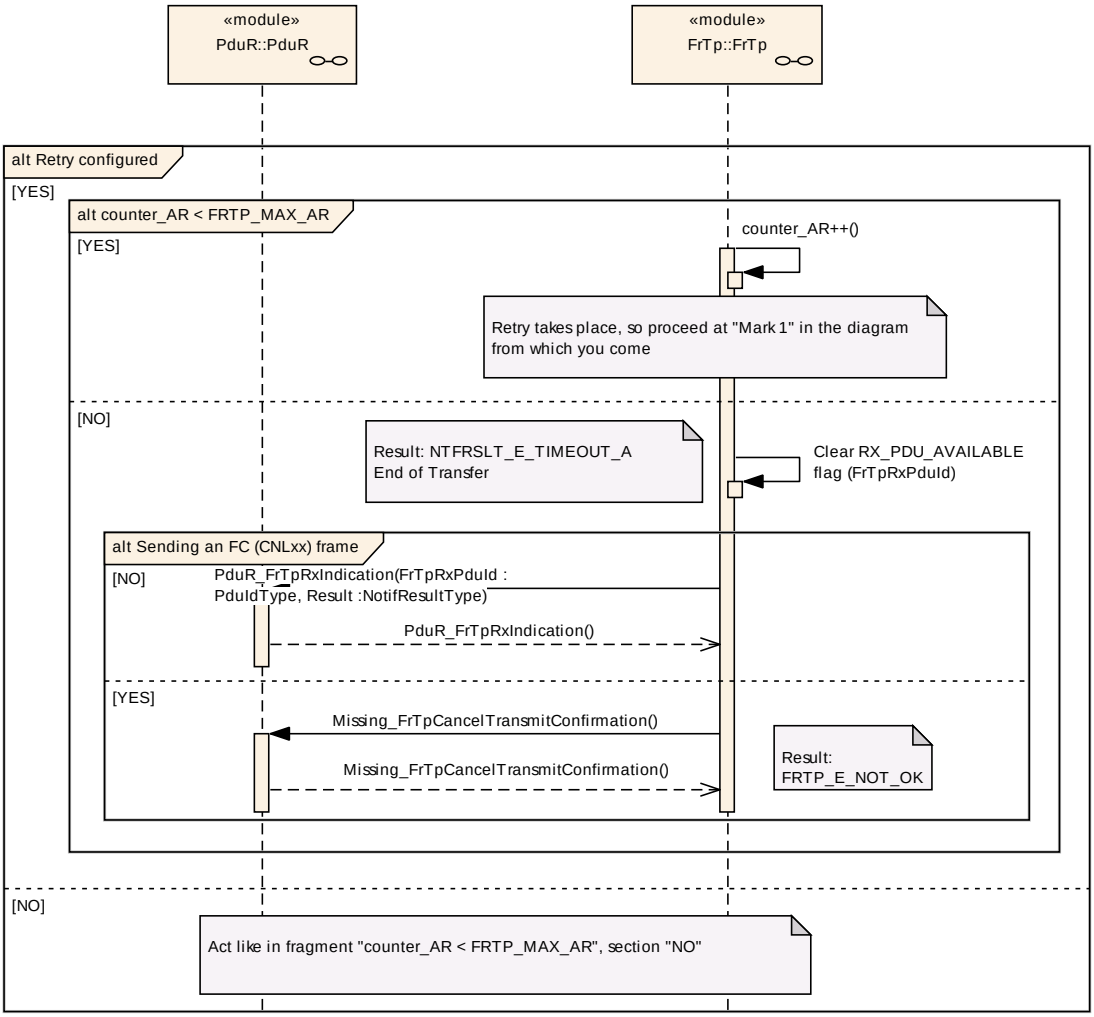
Comments:

9.2.2 Segmented Receiving



9.2.3 Others

9.2.3.1 Timeout AR



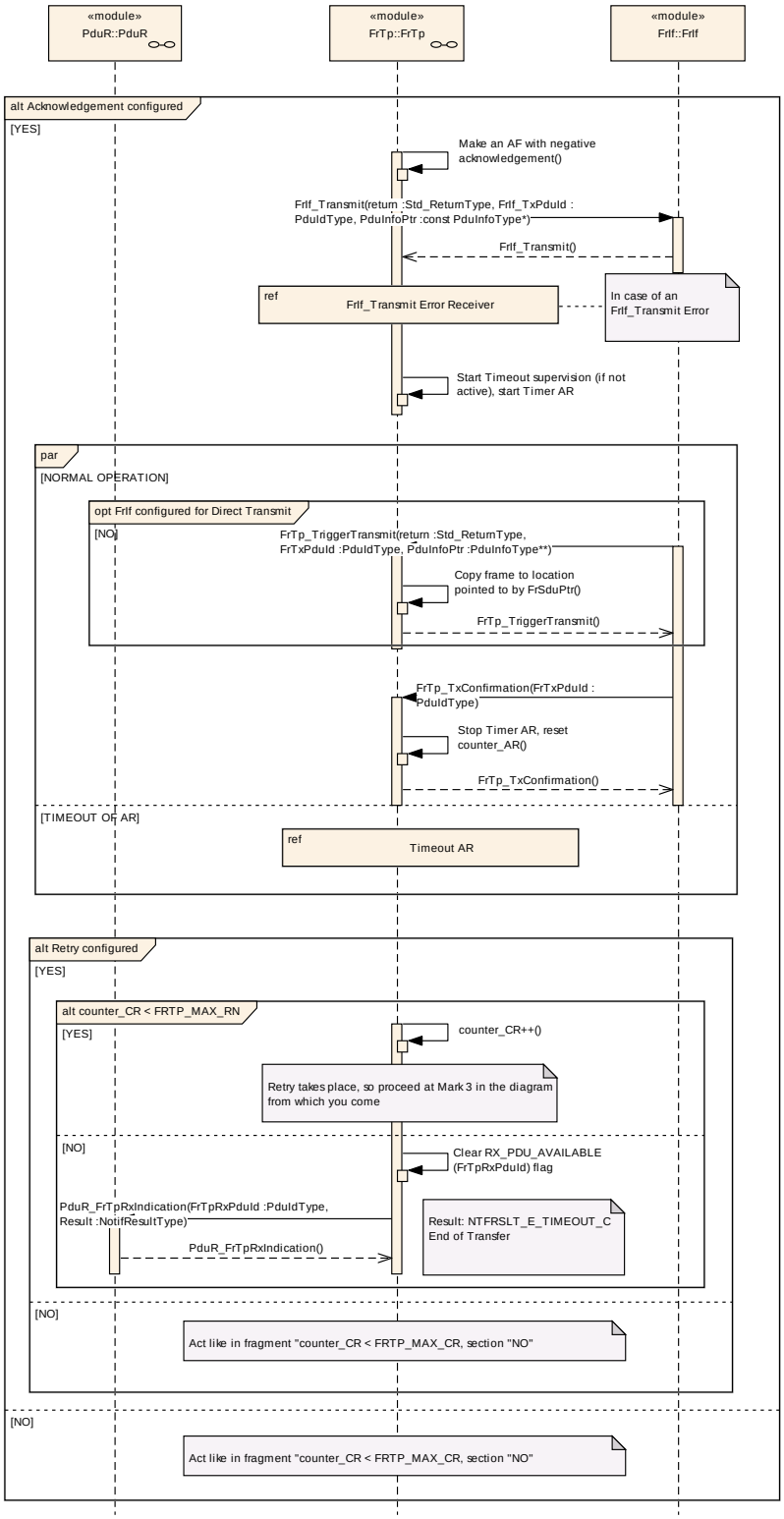
Status: POSTPONED TO AUTOSAR R2.1 !!!

Some finishing by TO for SWS 0.26
Details see "FlexRay UML Sequence Diagrams - Change History.doc"
2006-04-12: Update by BMW_TK to match FrIf SWS V1.2.5
2006-04-13: Update by DECOMSYS_TGAL to match FrIf SWS V1.2.5
2006-04-24: Minor changes and corrections by BMW_TK
2006-04-27: Adaptation to Fr SWS V1.0.15
2007-11-22: Minor changes by BMW_MZ

Description:

Comments:

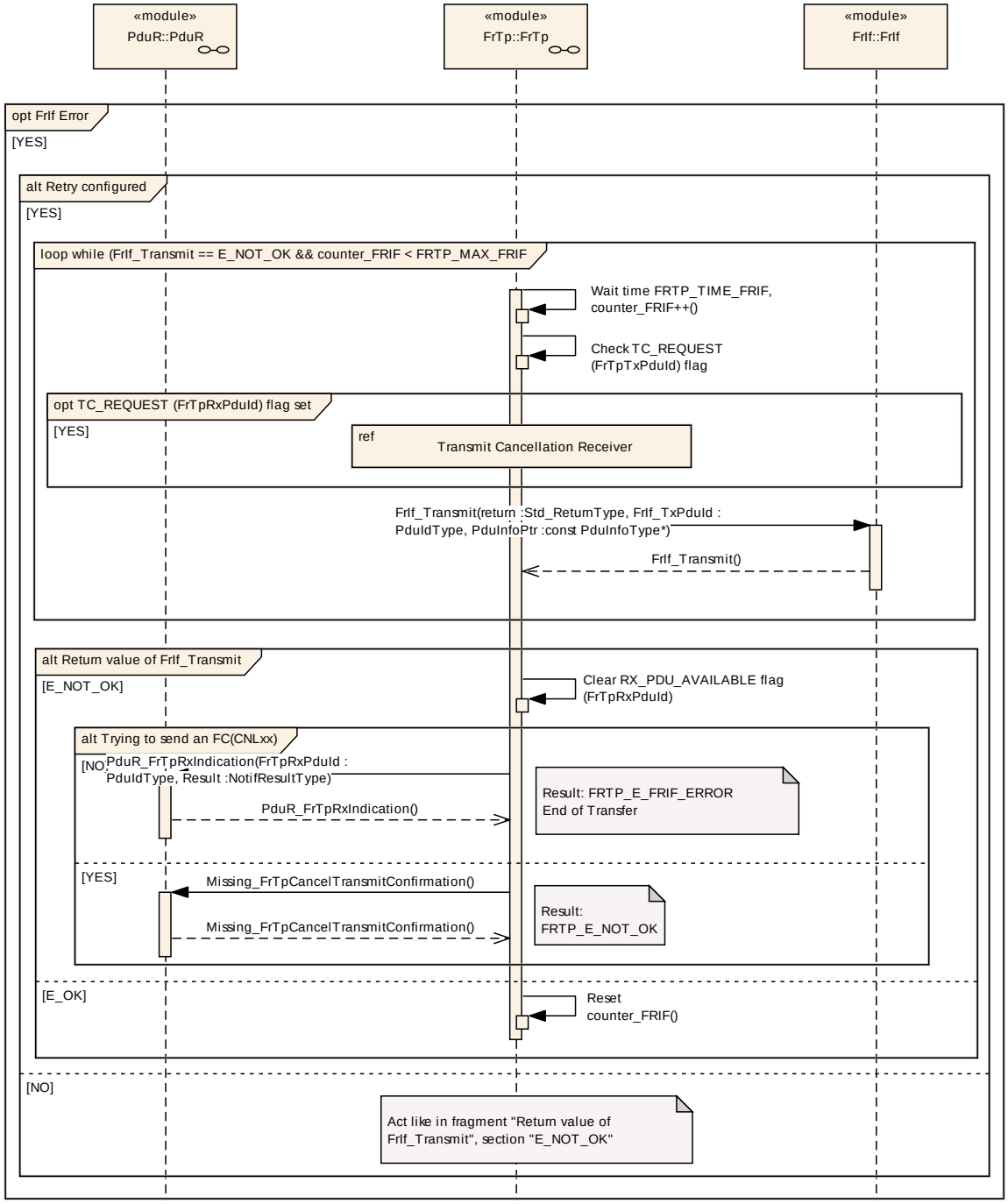
9.2.3.2 Timeout CR



Status: POSTPONED TO AUTOSAR R2.1 !!!

Some finishing by TO for SWS 0.26
Details see "FlexRay UML Sequence Diagrams - Change History.doc"
2006-04-12: Update by BMW_TK to match Frlf SWS V1.2.5
2006-04-13: Update by DECOMSYS_TGAL to match Frlf SWS V1.2.5
2006-04-24: Minor changes and corrections by BMW_TK
2006-04-27: Adaptation to Fr SWS V1.0.15
2007-11-22: Minor changes by BMW_MZ

9.2.3.3 FrIf_Transmit Error Receiver



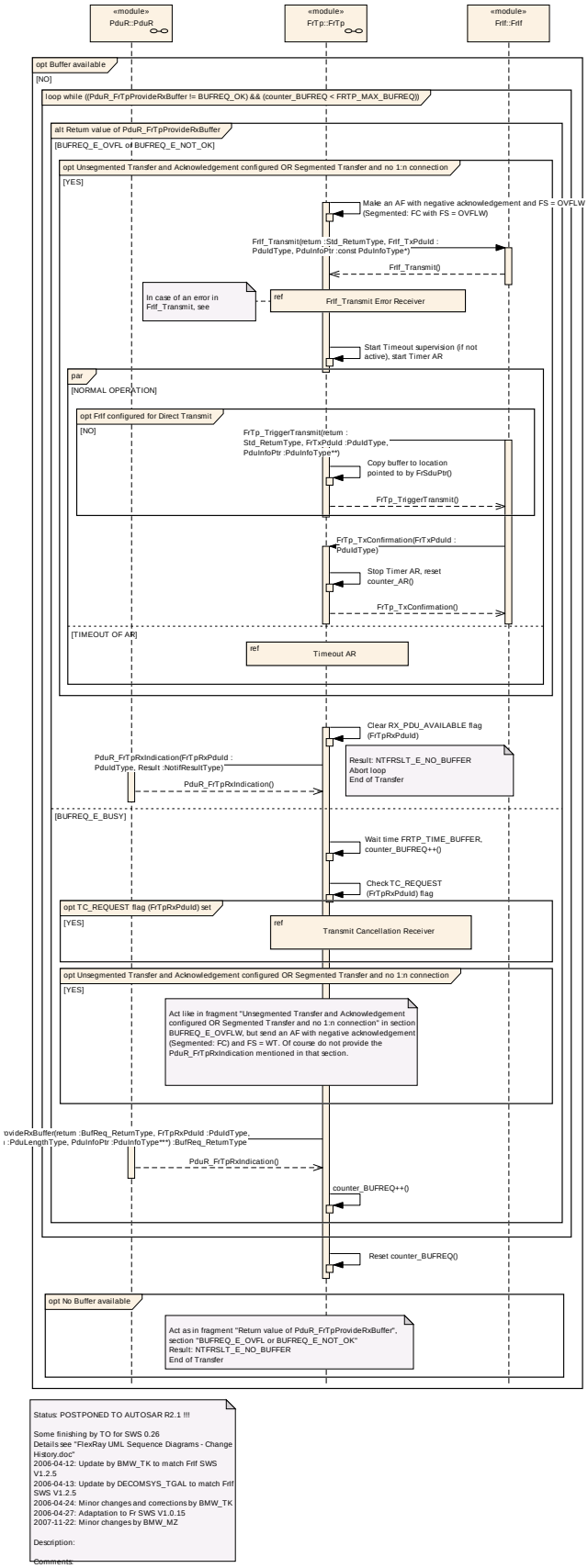
Status: POSTPONED TO AUTOSAR R2.1 !!!

Some finishing by TO for SWS 0.26
Details see "FlexRay UML Sequence Diagrams - Change History.doc"
2006-04-12: Update by BMW_TK to match FrIf SWS V1.2.5
2006-04-13: Update by DECOMSYS_TGAL to match FrIf SWS V1.2.5
2006-04-24: Minor changes and corrections by BMW_TK
2006-04-27: Adaptation to Fr SWS V1.0.15
2007-11-22: Minor changes by BMW_MZ

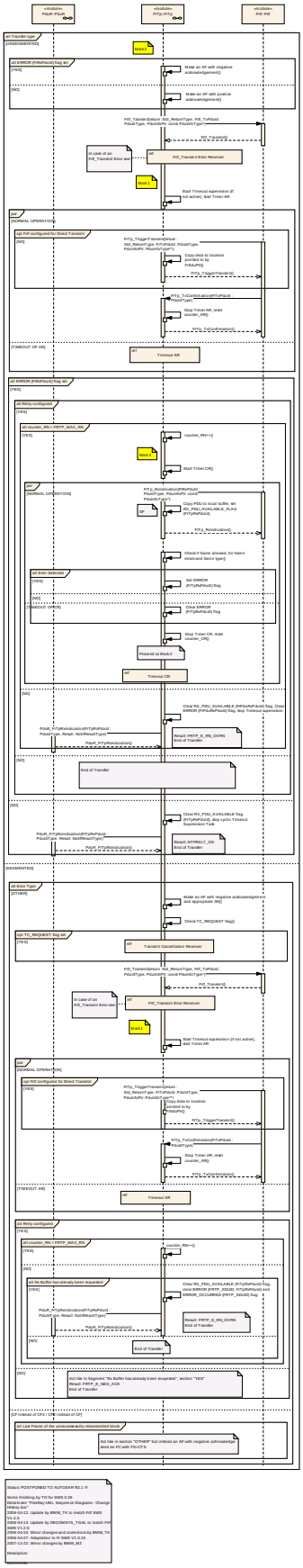
Description:

Comments:

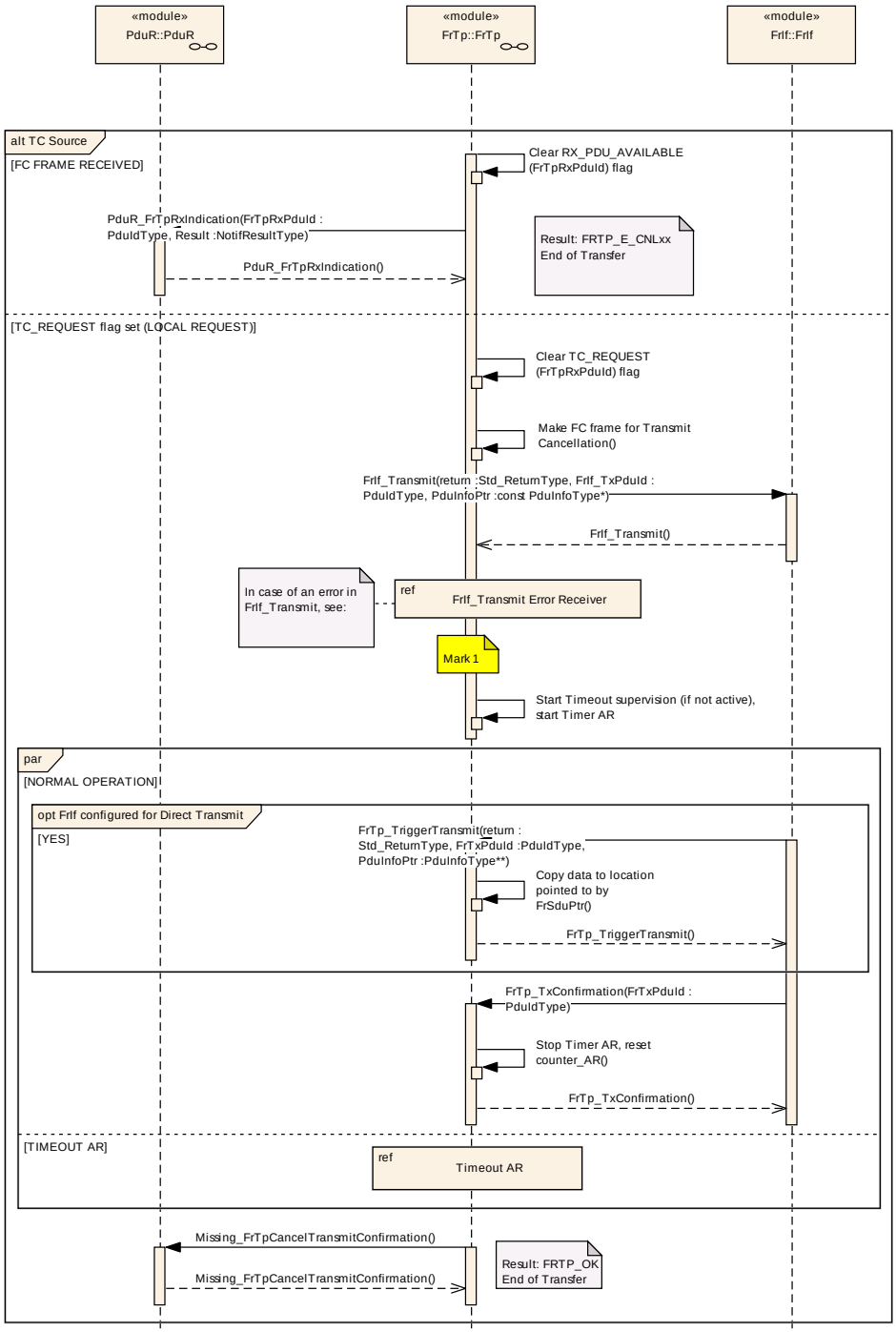
9.2.3.4 Buffer Request Error Receiver



9.2.3.5 Acknowledgement / Retry Receiver



9.2.3.6 Transmit Cancellation Receiver



Status: POSTPONED TO AUTOSAR R2.1 !!!

Some finishing by TO for SWS 0.26
Details see "FlexRay UML Sequence Diagrams - Change History.doc"
2006-04-12: Update by BMW_TK to match Frlf SWS V1.2.5
2006-04-13: Update by DECOMSYS_TGAL to match Frlf SWS V1.2.5
2006-04-24: Minor changes and corrections by BMW_TK
2006-04-27: Adaptation to Fr SWS V1.0.15
2007-11-22: Minor changes by BMW_MZ

Description:

Comments:

10 Configuration specification

FRTP199: In general, this chapter defines configuration parameters and their clustering into containers. In order to support the specification Chapter 10.1 describes fundamentals. It also specifies a template (table) you shall use for the parameter specification. We intend to leave Chapter 10.1 in the specification to guarantee comprehension.

Chapter 10.2 specifies the structure (containers) and the parameters of the module FlexRay Transport Layer.

Chapter 10.3 specifies published information of the module FlexRay Transport Layer

10.1 How to read this chapter

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

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

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

10.1.1 Configuration and configuration parameters

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

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

10.1.2 Variants

Yes

10.1.3 Containers

Containers structure the set of configuration parameters. This means:

- *all* configuration parameters are kept in containers.

- (sub-) containers can reference (sub-) containers. It is possible to assign a multiplicity to these references. The multiplicity then defines the possible number of instances of the contained parameters.

10.1.4 Specification template for configuration parameters

The following tables consist of three sections:
- the general section
- the configuration parameter section
- the section of included/referenced containers

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

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

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

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

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

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

10.2 Containers and configuration parameters

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

The following picture gives an overview about the configuration.

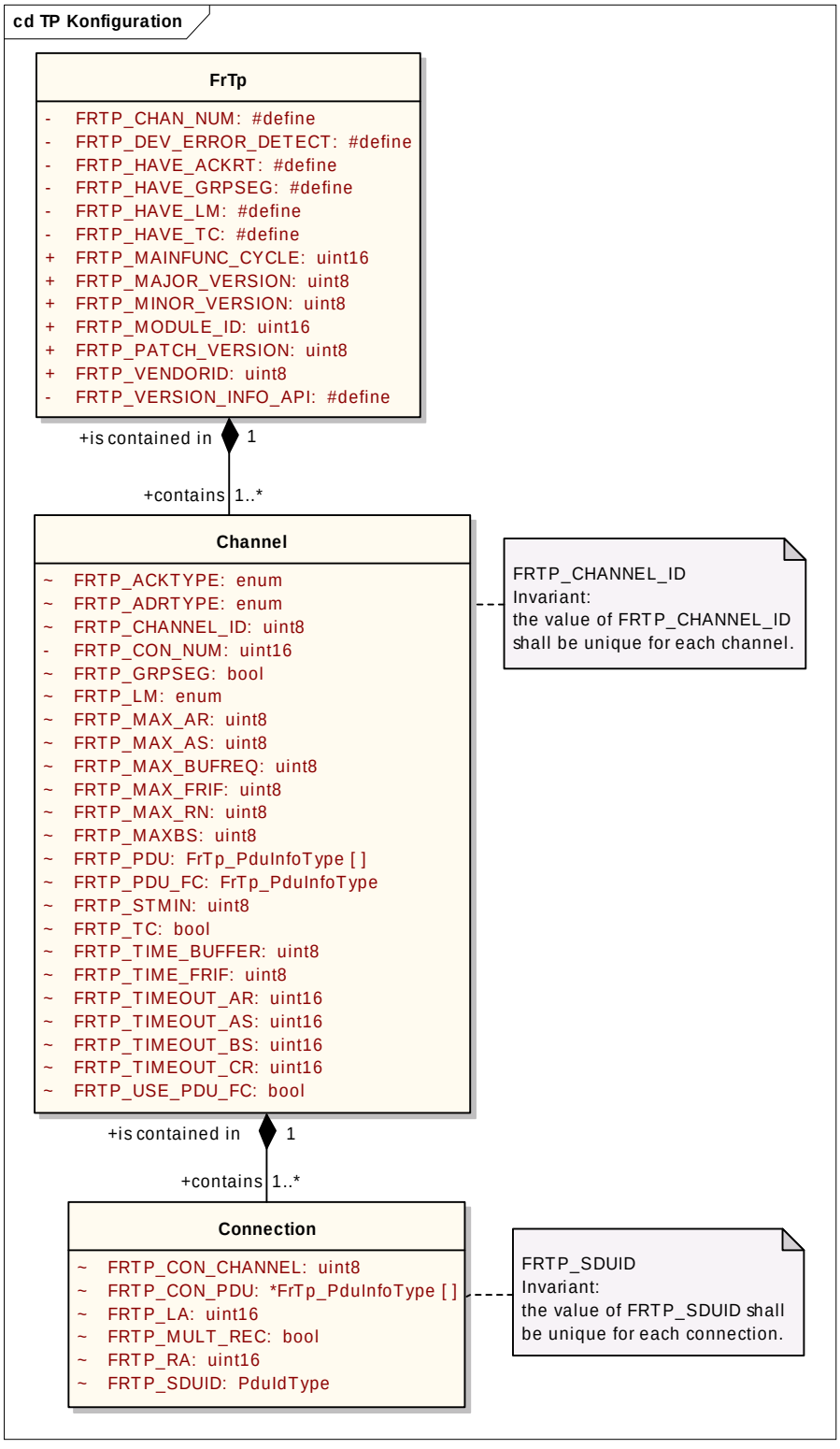


Figure 23: Overview over FrTp configuration

10.2.1 Variants

Variant 1: Pre Compile time

Variant 2: Mixture of Pre Compile time and Post Build Time Parameters

10.2.2 FrTp

Module Name	*FrTp*
Module Description	Configuration of the FrTp (FlexRay Transport Protocol) module.

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
FrTpGeneral	1	This container contains the general configuration (parameters) of the FlexRay TP.
FrTpMultipleConfig	1	This container holds one or several multiple configuration sets.

10.2.3 FrTpGeneral

SWS Item	**--**
Container Name	FrTpGeneral
Description	This container contains the general configuration (parameters) of the FlexRay TP.
Configuration Parameters	

SWS Item	**--**		
Name	FrTpChanNum {FRTP_CHAN_NUM}		
Description	Preprocessor switch for defining the number of concurrent channels the module supports. Up to 32 channels shall be definable here.		
Multiplicity	1		
Type	IntegerParamDef		
Range	1 .. 32		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**--**		
Name	FrTpDevErrorDetect {FRTP_DEV_ERROR_DETECT}		
Description	Preprocessor switch for enabling development error detection.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**--**
Name	FrTpHaveAckRt {FRTP_HAVE_ACKRT}
Description	Preprocessor switch for enabling the Acknowledgement and retry mechanisms.
Multiplicity	1
Type	BooleanParamDef

Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**--**		
Name	FrTpHaveGrpSeg {FRTP_HAVE_GRPSEG}		
Description	Preprocessor switch for enabling segmentation of 1:n messages.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**--**		
Name	FrTpHaveLm {FRTP_HAVE_LM}		
Description	Preprocessor switch for enabling the mechanism for message longer than allowed by.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**--**		
Name	FrTpHaveTc {FRTP_HAVE_TC}		
Description	Preprocessor switch for enabling Transmit Cancellation.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**--**		
Name	FrTpMainFuncCycle {FRTP_MAINFUNC_CYCLE}		
Description	This parameter contains the calling period of the TPs Main Function. The parameter is specified in seconds.		
Multiplicity	1		
Type	FloatParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	L	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**--**
Name	FrTpVersionInfoApi {FRTP_VERSION_INFO_API}
Description	Preprocessor switch for enabling the Version info API.
Multiplicity	1

Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

No Included Containers

FRTP186: All parameters within chapter 10.2.2 are global and, of course, only present once for the whole software module.

FRTP177: global configuration

10.2.4 FrTpChannel

SWS Item	**--**
Container Name	FrTpChannel{FrTpChannelConfiguration}
Description	This container contains the configuration (parameters) of one FlexRay TP channel.
Configuration Parameters	

SWS Item	**--**		
Name	FrTpAckType {FRTP_ACKTYPE}		
Description	This parameter defines the type of acknowledgement which is used for the specific channel.		
Multiplicity	1		
Type	EnumerationParamDef		
Range	FRTP_ACK_WITHOUT_RT	Acknowledgement without retry	
	FRTP_ACK_WITH_RT	Acknowledgement with retry	
	FRTP_NO	No acknowledgement	
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**--**		
Name	FrTpAdrType {FRTP_ADRTYPE}		
Description	This parameter states the addressing type this connection has. The meanings of the values are one byte and two byte.		
Multiplicity	1		
Type	EnumerationParamDef		
Range	FRTP_OB	One Byte	
	FRTP_TB	Two Bytes	
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**--**
Name	FrTpChannelId {FRTP_CHANNEL_ID}
Description	The Id of the channel.
Multiplicity	1
Type	IntegerParamDef (Symbolic Name generated for this parameter)

Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**--**		
Name	FrTpConNum {FRTP_CON_NUM}		
Description	This parameter states the number of connections used in this channel. At least 256 shall be configurable here.		
Multiplicity	1		
Type	IntegerParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**FRTP_GRPSEG :**		
Name	FrTpGrpSeg		
Description	Here can be specified, whether segmentation within a 1:n connection is allowed or not.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**--**		
Name	FrTpLm {FRTP_LM}		
Description	This specifies the maximum message length for the particular channel.		
Multiplicity	1		
Type	EnumerationParamDef		
Range	FRTP_ISO	Up to (2**12)-1 Byte message length (No FF-Ex or SF-E or AF shall be used and recognized)	
	FRTP_ISO6	As ISO, but the maximum payload length is limited to 6 byte (SF-I, FF-I, CF). This is necessary to route TP on CAN when using Extended Addressing or Mixed Addressing on CAN.	
	FRTP_L4G	SF-E allowed (SF of arbitrary length depending on FrTpPduLength), up to (2**32)-1 byte message length (all FF-x allowed).	
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**--**	
Name	FrTpMaxAr {FRTP_MAX_AR}	
Description	This parameter defines the maximum number of trying to send a frame when a TIMEOUT AR occurs (depending on whether retry is configured).	
Multiplicity	1	
Type	IntegerParamDef	
Range	0 .. 255	

Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**--**		
Name	FrTpMaxAs {FRTP_MAX_AS}		
Description	This parameter defines the maximum number of trying t osend a frame when a TIMEOUT AS occurs (depending on whether retry is configured)		
Multiplicity	1		
Type	IntegerParamDef		
Range	0 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**--**		
Name	FrTpMaxBs {FRTP_MAXBS}		
Description	This parameter is only relevant when having retry activated. It limits the maximal block size the FrTp can choose in order to limit the amount of Tx buffer that will be requested at the sender side in a segmented transfer.		
Multiplicity	1		
Type	IntegerParamDef		
Range	1 .. 16		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**--**		
Name	FrTpMaxBufReq {FRTP_MAX_BUFREQ}		
Description	This parameter defines the maximum number of trying to get a buffer (Transmit / Receive), depending of the return value of PduR_FrTpProvideTxBuffer / PduR_FrTpProvideRxBuffer and on whether retry is configured.		
Multiplicity	1		
Type	IntegerParamDef		
Range	0 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**--**		
Name	FrTpMaxFrIf {FRTP_MAX_FRIF}		
Description	This parameter defines the maximum number of trying to send a frame when the FrIf returns an error.		
Multiplicity	1		
Type	IntegerParamDef		
Range	0 .. 255		
Default value	--		

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

SWS Item	**--**		
Name	FrTpMaxRn {FRTP_MAX_RN}		
Description	This parameter defines the maximum number of retries (if retry is configured for the particular channel).		
Multiplicity	1		
Type	IntegerParamDef		
Range	0 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**--**		
Name	FrTpStMin {FRTP_STMIN}		
Description	This parameter defines the minimum amount of time between two succeeding CFs. Specified in seconds.		
Multiplicity	1		
Type	FloatParamDef		
Range	0.0 .. 255.0		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**FRTP_TC :**		
Name	FrTpTc		
Description	With this switch Transmit Cancellation can be turned on or off for this channel.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**--**		
Name	FrTpTimeBr {FRTP_TIME_BR}		
Description	This parameter defines the time in seconds between receiving the last CF of a block or an FF-x (or SF-x) and sending out an FC or AF. It is obvious that FRTP_TIME_BR + FRTP_TIMEOUT_AR < FRTP_TIMEOUT_BS must hold (because the transmission duration on the bus has also to be considered). This parameter is defined in ISO 15765-2. It is contained in the configuration as a performance requirement.		
Multiplicity	1		
Type	FloatParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants

	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**--**		
Name	FrTpTimeBuffer {FRTP_TIME_BUFFER}		
Description	This parameter defines the time in seconds of waiting for the next try (if retry is activated) to get a Tx or Rx buffer.		
Multiplicity	1		
Type	FloatParamDef		
Range	0.0 .. 0.255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**--**		
Name	FrTpTimeCs {FRTP_TIME_CS}		
Description	This parameter defines the time in seconds between the sending of two consecutive CFs or between a CF and a FC (for Transmit Cancellation) or between reception of an FC or AF and sending of the next CF or a FC (for Transmit Cancellation). It is obvious that FRTP_TIME_CS + FRTP_TIMEOUT_AS < FRTP_TIMEOUT_CR must hold (because the transmission duration on the bus has also to be considered). This parameter is defined in ISO 15765-2. It is contained in the configuration as a performance requirement.		
Multiplicity	1		
Type	FloatParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**--**		
Name	FrTpTimeFrIf {FRTP_TIME_FRIF}		
Description	This parameter defines the time in seconds of waiting for the next try (if retry is activated) to send via FrIf_Transmit.		
Multiplicity	1		
Type	FloatParamDef		
Range	0.0 .. 0.255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**--**
Name	FrTpTimeoutAr {FRTP_TIMEOUT_AR}
Description	This parameter states the timeout in seconds between the PDU transmit request of the Transport Layer to the FlexRay Interface and the corresponding confirmation of the FlexRay Interface on the receiver side (for FC or AF).
Multiplicity	1
Type	FloatParamDef

Range	0.0 .. 65.535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**--**		
Name	FrTpTimeoutAs {FRTP_TIMEOUT_AS}		
Description	This parameter states the timeout in seconds between the PDU transmit request for the first PDU of the group used in the current connection of the Transport Layer to the FlexRay Interface and the corresponding confirmation of the FlexRay Interface (when having sent the last PDU of the group used in this connection) on the sender side (SF-x, FF-x, CF or FC (in case of Transmit Cancellation)).		
Multiplicity	1		
Type	FloatParamDef		
Range	0.0 .. 65.535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**--**		
Name	FrTpTimeoutBs {FRTP_TIMEOUT_BS}		
Description	This parameter defines the timeout in seconds for waiting for an FC or AF on the sender side in a 1:1 connection.		
Multiplicity	1		
Type	FloatParamDef		
Range	0.0 .. 65.535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**--**		
Name	FrTpTimeoutCr {FRTP_TIMEOUT_CR}		
Description	This parameter defines the timeout value in seconds for waiting for a CF or FF-x (in case of retry) after receiving the last CF or after sending an FC or AF on the receiver side.		
Multiplicity	1		
Type	FloatParamDef		
Range	0.0 .. 65.535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**--**
Name	FrTpUsePduFc {FRTP_USE_PDU_FC}
Description	This switch defines, whether within this channel the dedicated FC/ACK PDU (FrTpPduFc) shall be used or not. If this is not used FC / ACK frames are sent using the normal IDs, otherwise only FrTpPduFc shall be used for

	sending / receiving FC / ACK frames.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
FrTpConnection	1..*	This container contains the configuration (parameters) of one FlexRay TP connection. A connection can only belong to one channel.
FrTpPdu	1..*	Container to hold the PDU parameters. ImplementationType: FrTp_PduInfoType
FrTpPduFc	0..2	This is the identifier of the FlexRay Interface PDUs (Fr N-PDU, Fr L-SDU) in which the Transport Layer Flow Control and Acknowledgement Frames of this channel should be transmitted. ImplementationType: FrTp_PduInfoType

FRTP166: All parameters within this chapter are present for each channel and read-only. They shall be placed outside the source code of the module in order to be modifiable by a flashing process without re-flashing the code itself...

The parameters marked with a "*" are only relevant if the corresponding compile switch (FRTP_HAVE_...) is set to YES.

FRTP183:

Performance Requirements according to [10]
The two parameters, FRTP_TIME_BR and FRTP_TIME_CS, are **not software configuration parameters**, they are contained in [10] as performance requirements. They are just for information.

10.2.5 FrTpPdu

SWS Item	**--**
Container Name	FrTpPdu
Description	Container to hold the PDU parameters. ImplementationType: FrTp_PduInfoType
Configuration Parameters	

SWS Item	**--**		
Name	FrTpPduDirection		
Description	This parameter defines the direction of the PDU.		
Multiplicity	1		
Type	EnumerationParamDef		
Range	FRTP_RX	Received PDU	
	FRTP_TX	Transmitted PDU	
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE

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

SWS Item	**--**		
Name	FrTpPduId {FRTP_PDU}		
Description	This is the identifier of the FlexRay Interface PDUs (Fr N-PDU, Fr L-SDU) in which the Transport Layer Frames of this channel should be transmitted. ImplementationType: PduIdType		
Multiplicity	1		
Type	IntegerParamDef (Symbolic Name generated for this parameter)		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**--**		
Name	FrTpPduRef		
Description			
Multiplicity	1		
Type	Reference to Pdu		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

No Included Containers

10.2.6 FrTpPduFc

SWS Item	**--**
Container Name	FrTpPduFc
Description	This is the identifier of the FlexRay Interface PDUs (Fr N-PDU, Fr L-SDU) in which the Transport Layer Flow Control and Acknowledgement Frames of this channel should be transmitted. ImplementationType: FrTp_PduInfoType
Configuration Parameters	

SWS Item	**--**		
Name	FrTpPduFcDirection		
Description	This parameter defines the direction of the PDU.		
Multiplicity	1		
Type	EnumerationParamDef		
Range	FRTP_FC_RX	Received flow control PDU	
	FRTP_FC_TX	Transmitted flow control PDU	
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	**--**
Name	FrTpPduFcId {FRTP_PDU_FC}
Description	This is the identifier of the FlexRay Interface PDUs (Fr N-PDU, Fr L-SDU) in which the Transport Layer Flow Control and Acknowledgement Frames

	of this channel should be transmitted.		
Multiplicity	1		
Type	IntegerParamDef (Symbolic Name generated for this parameter)		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**--**		
Name	FrTpPduFcRef		
Description			
Multiplicity	1		
Type	Reference to Pdu		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

No Included Containers

10.2.7 FrTpConnection

SWS Item	**--**
Container Name	FrTpConnection{FrTpConnectionConfiguration}
Description	This container contains the configuration (parameters) of one FlexRay TP connection. A connection can only belong to one channel.
Configuration Parameters	

SWS Item	**--**		
Name	FrTpLa {FRTP_LA}		
Description	This parameter defines the Local Address for the respective connection. When the local instance is the sender, this is the Source Address within the TP frame. When the local instance is the receiver, this is the Target Address within the TP frame. Note that in case of 1 byte addressing only the values from 0x0000 - 0x00FF are valid.		
Multiplicity	1		
Type	IntegerParamDef		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**--**
Name	FrTpMultRec {FRTP_MULT_REC}
Description	This parameter defines, whether this connection is an 1:1 ('false') or an 1:n ('true') connection. Of course, if the channel to which the connection is configured has retry or acknowledgement enabled, no retry or acknowledgement will occur in case the connection is an 1:n connection.
Multiplicity	1
Type	BooleanParamDef

Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**--**		
Name	FrTpRa {FRTP_RA}		
Description	This parameter defines the Remote Address for the respective connection. When the local instance is the sender, this is the Target Address within the TP frame. When the local instance is the receiver, this is the Source Address within the TP frame. Note that in case of 1 byte addressing only the values from 0x0000 - 0x00FF are valid.		
Multiplicity	1		
Type	IntegerParamDef		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**--**		
Name	FrTpConPduRef {FRTP_CON_PDU}		
Description	Each value defines a PDU to be used for this connection. Thus each value is a PDU-ID given in FrTpPdu and this array cannot be longer than the array FrTpPdu. Please note: Only PDUs of the same size shall be used within a connection. Of course the PDU having the TxConfirmation configured has to be used by every connection.		
Multiplicity	1..*		
Type	Reference to FrTpPdu		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
FrTpRxSdu	0..1	Describes the Rx SDU
FrTpTxSdu	0..1	Describes the Tx SDU

FRTP185: All parameters within this chapter are present for each connection and read-only. They shall be placed outside the source code of the module in order to be modifiable by a flashing process without re-flashing the code itself.

FRTP168: Parameters

10.2.8 FrTpTxSdu

SWS Item	**--**
Container Name	FrTpTxSdu
Description	Describes the Tx SDU
Configuration Parameters	

SWS Item	**--**		
Name	FrTpSduTxId {FRTP_SDUID}		
Description	This is a unique identifier for a received or a to be transmitted message. With this (and by means of e.g. a lookup table) the PDU Router can route the message appropriately without dealing with the particularities of the Transport Layer. This parameter can also be seen as the identifier of a connection. ImplementationType: PduIdType		
Multiplicity	1		
Type	IntegerParamDef (Symbolic Name generated for this parameter)		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**--**		
Name	FrTpTxSduRef		
Description	Reference to a PDU in the global PDU structure.		
Multiplicity	1		
Type	Reference to Pdu		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

No Included Containers

10.2.9 FrTpRxSdu

SWS Item	**--**
Container Name	FrTpRxSdu
Description	Describes the Rx SDU
Configuration Parameters	

SWS Item	**--**		
Name	FrTpRxSduRef		
Description	Reference to a PDU in the global PDU structure.		
Multiplicity	1		
Type	Reference to Pdu		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

No Included Containers

10.2.10 FrTpMultipleConfig

SWS Item	**--**
Container Name	FrTpMultipleConfig [Multi Config Container]
Description	This container holds one or several multiple configuration sets.
Configuration Parameters	

Included Containers

Container Name	*Multiplicity*	*Scope / Dependency*
FrTpChannel	1..*	This container contains the configuration (parameters) of one FlexRay TP channel.

10.3 Published Information

FRTP178: published parameters

The standard common published information like

vendorId (<Module>_VENDOR_ID),
moduleId (<Module>_MODULE_ID),
arMajorVersion (<Module>_AR_MAJOR_VERSION),
arMinorVersion (<Module>_ AR_MINOR_VERSION),
arPatchVersion (<Module>_ AR_PATCH_VERSION),
swMajorVersion (<Module>_SW_MAJOR_VERSION),
swMinorVersion (<Module>_ SW_MINOR_VERSION),
swPatchVersion (<Module>_ SW_PATCH_VERSION),
vendorApiInfix (<Module>_VENDOR_API_INFIX)

is provided in the BSW Module Description Template (see [9] Figure 4.1 and Figure 7.1).

Additional published parameters are listed below if applicable for this module.

10.4 Important Issues on Configuration

10.4.1 Start and Stop of the Timing Parameters of Chapter 10.2.3

FRTP169: The table below gives an overview when the time of each of these parameters start to run and when it is stopped. Note that if SF-x is mentioned it is meant in the case acknowledgement is configured (the same for AF).

FRTP170: For 1:n connections only the parameters `FRTP_TIMEOUT_AS`, `FRTP_TIME_CS` (only CF) and `FRTP_TIMEOUT_CR` (only CF) hold, since no flow control or acknowledgement is allowed in that case.

Timing Parameter	*Start*	*Stop*
`FRTP_TIMEOUT_AS`	*FrIf_Transmit* (first PDU of the group used by the current connection)	*FrTp_TxConfirmation* (for the last PDU of the group used by the current connection)
`FRTP_TIMEOUT_AR`	*FrIf_Transmit*	*FrTp_TxConfirmation*
`FRTP_TIMEOUT_BS`	*FrTp_TxConfirmation* (SF-x, FF-x or last CF of a block), *FrTp_RxIndication* (FC or AF,	*FrTp_RxIndication* (FC or AF)

	both in case of FR_FS = WAIT)	
FRTP_TIME_BR	*FrTp_RxIndication* (FF-x, last CF of a block or SF-x), *FrTp_TxConfirmation (FC or AF, both in case of FR_FS = WAIT)*	*FrIf_Transmit* (FC or AF)
FRTP_TIMEOUT_CR	*FrTp_RxIndication* (CF), *FrTp_TxConfirmation* (FC or AF)	*FrTp_RxIndication* (CF or SF-x, FF-x (the latter two in case of retry))
FRTP_TIME_CS	*FrTp_TxConfirmation* (CF), *FrTp_RxIndication* (FC or AF (not after the last one))	*FrIf_Transmit* (CF, FC for Transmit Cancellation)

Table 3: Start and Stop of the different timeouts and times

10.4.2 How to get an ISO compliant Channel / Connection

FRTP171: To achieve ISO compliance within a channel/connection, there are restrictions for some parameters. Those marked with a "*" are only relevant, if the features are compiled in (see chapter 10.2.2).

These and those are explained in the table below:

Parameter	*Allowed values*
FRTP_ACKTYPE (*)	'FRTP_NO'
FRTP_GRPSEG (*)	false
FRTP_TC (*)	false
FRTP_LM (*)	'FRTP_ISO', 'FRTP_ISO6'
FRTP_PDU_LENGTH	9 [FRTP_ADRTYPE == FRTP_OB, FRTP_LM == FRTP_ISO6], 10 [FRTP_ADRTYPE == FRTP_OB, FRTP_LM == FRTP_ISO], 11 [FRTP_ADRTYPE == FRTP_TB, FRTP_LM == FRTP_ISO6], 12 [FRTP_ADRTYPE == FRTP_TB, FRTP_LM == FRTP_ISO]

Table 4: Parameter Setting for ISO compliance

All not mentioned parameters can have arbitrary values.

10.4.3 Dependencies among the Parameters

FRTP172: There are several dependencies among the connection specific and channel specific configuration parameters:

- If FRTP_MULT_REC sets the connection to be a 1:1 connection, then the value of FRTP_GRPSEG does not play a role for this connection since it is only relevant for 1:n connections.
- If FRTP_MULT_REC sets the connection to be a 1:n connection, then the values of FRTP_ACKTYPE, FRTP_MAXBS and FRTP_MAX_RN do not play a role for this connection since they are only relevant for 1:1 connections.
- If FRTP_MULT_REC sets the connection to be a 1:n connection or FRTP_ACK does not activate retry (FRTP_NO, FRTP_ACK_WITHOUT_RT) then the value of FRTP_MAXBS does not play a role for this connections since it is only relevant in 1:1 connections within channels with retry being activated.

10.4.4 Timing Constraints

The following Constraints shall hold for the Timing parameters:

1. V_E + FRTP_TIME_BR + (FRTP_TIMEOUT_AR * FRTP_MAX_AR) + V_S < FRTP_TIMEOUT_BS
2. V_S + FRTP_TIME_CS + (FRTP_TIMEOUT_AS * FRTP_MAX_AS)+ V_E < FRTP_TIMEOUT_CR

Where V_E is the time from Starting the BS Timer until recognition of the frame in the receiver TP and V_S is the time from Starting the CR Timer until recognition of the frame in the sender TP.

If retry is enabled, the following constraint should hold, too:

FRTP_TIMEOUT_BS + (FRTP_TIMEOUT_AS * FRTP_MAX_AS) + V_E < FRTP_TIMEOUT_CR

10.4.5 Configuration Requirements on the FlexRay Transport Layer

FRTP173: Both the parameter FRTP_MAX_BUFREQ and FRTP_MAX_RN have to have the same value in the sender and the receiver peer. This is necessary because they manifest in bus communication. So it can be avoided waiting for another FC(WT) or retry at the receiver side or doing additional ones at the sender side.

FRTP180: It has to be assured, that FRTP_STMIN < FRTP_TIMEOUT_CR since there will always be a timeout of the latter one otherwise.

FRTP181: The configuration of a connection and a channel shall be, of course, the same at the sender and the receiver side. Only the values of FRTP_LA and FRTP_RA are swapped.

10.4.6 Configuration Requirements on the FlexRay Interface

FRTP174: If more than one Fr N-PDU is used for one Fr N-SDU within a connection, the FrIf shall guarantee, that the Fr N-PDUs (Fr L-SDUs) are scheduled (sent over the bus) in the same order the FlexRay Transport Layer uses them, i. e. in ascending order regarding the Fr-N-PDU IDs used in the FlexRay Transport Layer. Furthermore these PDUs shall be scheduled with the same frequency and within one Job (concerning the Joblist) in the FlexRay Interface (since the reading of the PDU-Available Information for all PDUs of an connection has to be atomic.)
This is necessary to avoid CFs coming out of order in a segmented transfer.
FRTP175: For every FrTp L-SDU the PDU-Update/Valid Information of the FrIf shall be activated.
This is necessary to avoid Rx-Indication at the FrTp for in the current transfer not used Fr N-PDUs or if e. g. in every 2nd FlexRay bus cycle an Fr N-PDU is scheduled.
FRTP176: For every FrTp L-SDU no FrIf Trigger Transmit counter shall be utilized, i. e. the limit of the respective counter shall be 1. This is necessary in order to avoid multiple calls of *FrTp_TriggerTransmit* for the same Fr N-PDU in case e. g. a retry is necessary due to an timeout of the AS / AR timer.

FRTP182: For the group of FrTp L-SDUs used by a specific connection of the FrTp, a Tx Confirmation shall be configured in order to stop the AS / AR timer at the right point in time. This shall be done by configuring the Tx Confirmation to be given each time after sending of the last FrTp L-SDU (regarding to the order of using the FrTp_L-SDUs in the FrTp) of the group.
Since a connection can only use PDUs of the same length, for each group of PDUs used by a channel (where a group is identified by the length of its PDUs) exactly 1 PDU of each group shall have a TxConfirmation configured. This has to be the PDU with the highest ID within the respective group (because the FrTp uses the PDUs in ascending order)..

Example:
If a connection uses FrTp N-PDUs 1, 2, 3 and the message length requires 8 FrTp N-PDUs to be sent, then 1, 2 and 3 are sent, a TxConfirmation is given, again 1, 2, 3 is sent, again a TxConfirmation is given, 2 and 3 are sent and a TxConfirmation is given.

11 Changes to Release 1

11.1 Deleted SWS Items

SWS Item	*Rationale*
FRTP167	Bugfix
FRTP040	Update
FRTP041	Update
FRTP042	Update
FRTP043	Update
FRTP044	Update
FRTP045	Update
FRTP046	Update
FRTP047	Update
FRTP048	Update
FRTP049	Update
FRTP050	Update
FRTP051	Update
FRTP052	Update
FRTP053	Update

11.2 Replaced SWS Items

No replaced SWS Items to release 1.

11.3 Changed SWS Items

SWS Item	*Rationale*
FRTP009	Bugfix
FRTP010	Bugfix
FRTP019	Bugfix
FRTP020	Bugfix
FRTP086	Bugfix
FRTP090	Bugfix
FRTP091	Bugfix
FRTP093	Bugfix
FRTP097	Bugfix
FRTP098	Bugfix
FRTP148	Bugfix
FRTP168	Bugfix
FRTP171	Bugfix
FRTP172	Bugfix
FRTP174	Bugfix
FRTP177	Bugfix
FRTP178	Bugfix
FRTP182	Bugfix
FRTP183	Bugfix
FRTP184	Bugfix
FRTP185	Bugfix

FRTP186	Bugfix
FRTP188	Bugfix
FRTP189	Bugfix
FRTP190	Bugfix
FRTP030	Update
FRTP055	Update
FRTP060	Update
FRTP073	Update
FRTP084	Update
FRTP183	Update, Bugfix

11.4 Added SWS Items

SWS Item	***Rationale***
FRTP191	Bugfix
FRTP192	Bugfix
FRTP193	Bugfix
FRTP194	Bugfix
FRTP195	Bugfix
FRTP196	Bugfix
FRTP197	Bugfix
FRTP198	Bugfix
FRTP199	Bugfix
FRTP200	Bugfix
FRTP201	Bugfix
FRTP202	Bugfix
FRTP203	Bugfix
FRTP204	Bugfix
FRTP205	Bugfix
FRTP206	Bugfix
FRTP207	Bugfix
FRTP208	Bugfix
FRTP209	Bugfix
FRTP210	Bugfix
FRTP211	Bugfix
FRTP212	Bugfix
FRTP213	Bugfix
FRTP214	Bugfix
FRTP215	Bugfix
FRTP216	Bugfix
FRTP217	Bugfix
FRTP218	Bugfix

12 Changes during SWS Improvements by Technical Office

12.1 Deleted SWS Items

None

12.2 Replaced SWS Items

None

12.3 Changed SWS Items

None

12.4 Added SWS Items

SWS Item	*Rationale*
FRTP219	UML model linking of the mandatory interfaces
FRTP220	UML model linking of the optional interfaces