Document Title	Specification of FlexRay ISO Transport Layer
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	589
Document Classification	Standard

Document Version	1.2.0
Document Status	Final
Part of Release	3.2
Revision	3

Document Change History

Date	Version	Changed by	Change Description
28.02.2014	1.2.0	AUTOSAR Release Management	• Avoided the retry mechanism to get a buffer from PDUR. • Added new requirement describing the layout of BC parameter. • Modified FRISOTP1160 • Removed FRISOTP1068, FRISOTP1136, FRISOTP015_Conf, FRISOTP030_Conf, FRISOTP173 • Editorial changes
17.05.2012	1.1.1	AUTOSAR Administration	• Removed NTFRSLT_E_CANCELATION_OK from NotifResultType • Removed references to ChangeParameterConfirmation • Removed private values in NotifResultType
27.04.2011	1.0.0	AUTOSAR Administration	Initial Release (based on R4.0 FlexRay Transport Layer)

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, API, and the configuration of the AUTOSAR basic software module FlexRay ISO Transport Protocol (FrIsoTp) based on ISO 10681-2 standard.

According to the AUTOSAR layered software architecture [2] (see Figure 1), the FlexRay ISO Transport Protocol (FrIsoTp) is placed between the PDU Router module and the FlexRay Interface module. The main purpose of the FlexRay ISO Transport Protocol is segmentation and reassembly of messages (I-PDUs) that do not fit in one of the assigned FlexRay L-PDUs.

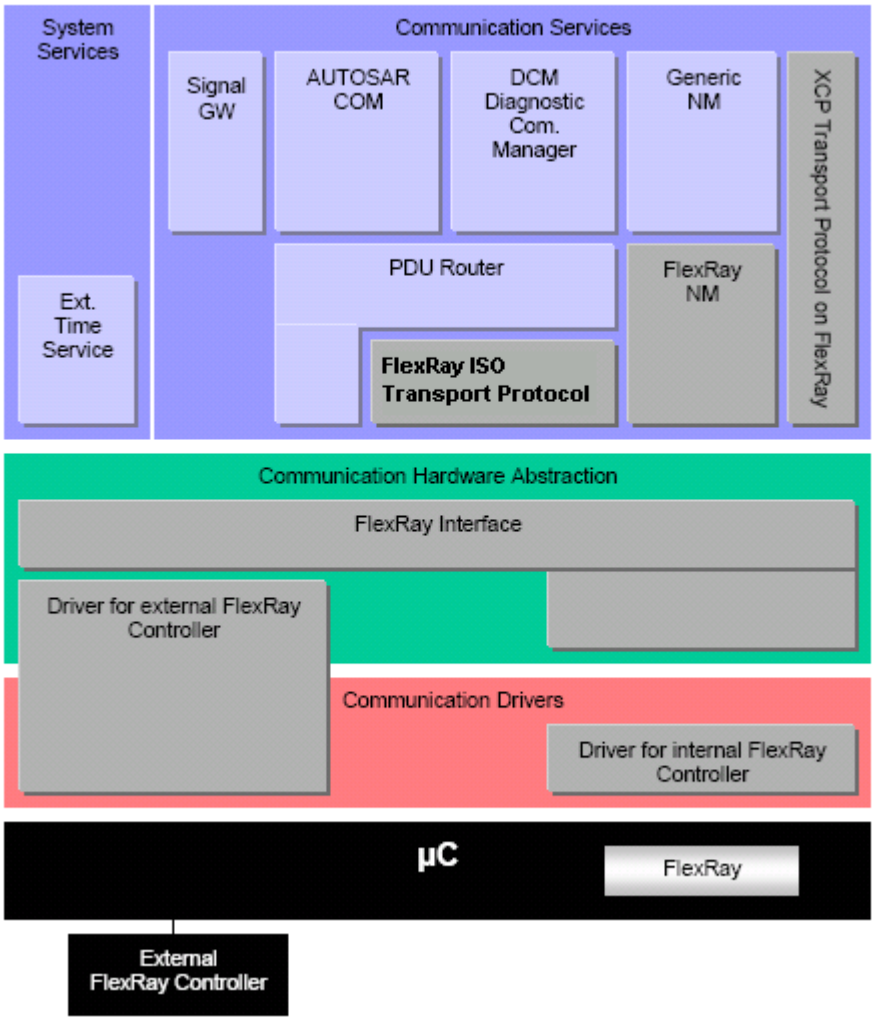


Figure 1: AUTOSAR FlexRay Layered Architecture

Figure 2 depicts the different PDU names in AUTOSAR nomenclature and the signal dependencies between the different AUTOSAR modules.

The PDU Router deploys upper Layers (e.g. COM, DCM etc) I-PDUs onto different communication protocols. The routing through a network system type (e.g. FlexRay, CAN, LIN etc.) depends on the I-PDU identifier. The PDU Router also determines (by

configuration) if a transport protocol shall be used or not. Finally, this module carries out gateway functionality, when there is no rate conversion.

FlexRay Interface (FrIf) provides standardized mechanisms to access a FlexRay bus channel via a FlexRay Communication Controller regardless of its location (µC internal/external). Depending on the PDU ID, the FlexRay Interface has to forward an N-PDU to FrIsoTp or an I-PDU to PduR. The FrIsoTp handles only transport protocol N-PDUs (i.e. SF, CF, LF and FC PDUs).

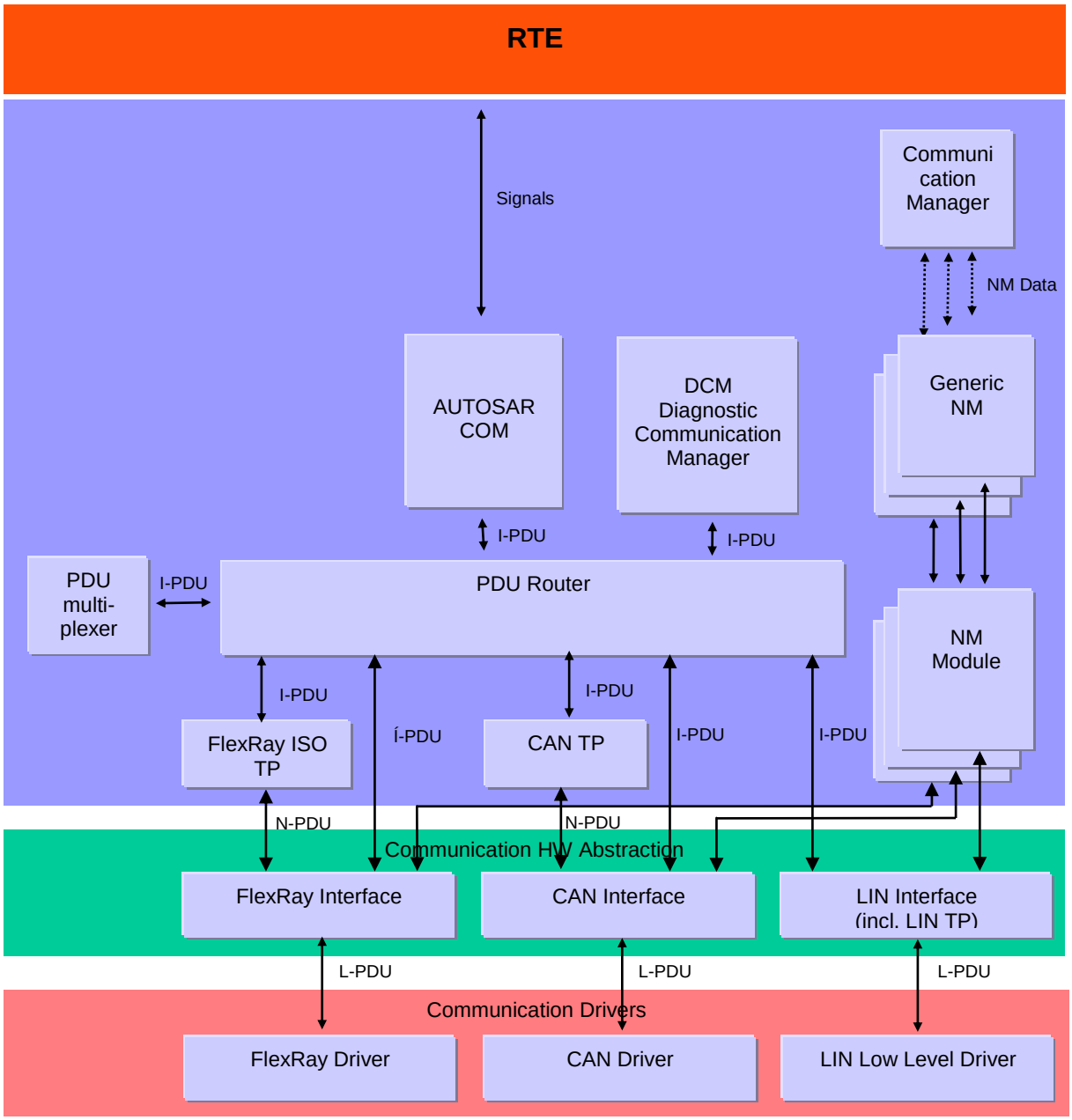


Figure 2 : AUTOSAR Signal Nomenclature

The main purpose of FlexRay ISO Transport Protocol is to perform a transfer of a message (I-PDU) that e.g. might or might not fit in a single FlexRay L-PDU. I-PDUs that do not fit into a single FlexRay L-PDU are segmented into multiple parts, where

each can be transmitted in a FlexRay L-PDU. According to AUTOSAR basic software architecture, the FlexRay ISO Transport Protocol provides methods for

- Segmentation of data in transmit direction
- Reassembling of data in receive direction
- Control of data flow
- Error detection in segmentation sessions
- Transmit cancellation

It is an AUTOSAR decision to base basic software module specifications on existing standards, thus this AUTOSAR FlexRay ISO Transport Layer specification is based on the international standard ISO 10681-2 [16]. This standard provides

- Transmission of a message with known message length
- Transmission of a message with unknown but finite message length
- Acknowledgement of transmission with retry mechanism

This specification supports only ISO 10681-2 compatible FlexRay Transport Protocol. Since the Release 4.0, the FlexRay is compatible to ISO 10681-2 specification, the same is being added in Release 3.2 to give a backward compatibility. The specification [14] supports only AUTOSAR compatible FlexRay Transport Protocol. User must be cautious in choosing which specification is used for FlexRay Transport Layer.

The FlexRay ISO Transport Protocol supports 1:1 and 1:n connections.

The FlexRay ISO Transport Protocol supports data transfers of up to 2^{16}-1 Bytes payload.

FlexRay ISO Transport Protocol is mainly used for vehicle diagnostic systems. Nevertheless, it was developed to deal with requirements from other FlexRay based systems requiring a Transport Protocol Layer protocol (e.g. Diagnostic communication, Inter-ECU communication, XCP communication etc.).

2 Acronyms and abbreviations

The prefix notation used in this document, is as follows:

Prefix:	*Description:*
I-	Relative to upper AUTOSAR Layer (e.g. COM, DCM etc.)
L-	Relative to the FlexRay Interface module.
N-	Relative to the FlexRay ISO Transport Protocol Layer.

All acronyms and abbreviations, which are specific to the FlexRay ISO Transport Layer and are therefore not contained in the AUTOSAR glossary are described in the following:

Acronym:	*Description:*
Fr L-SDU	This is the SDU of the FlexRay Interface module. It is similar to Fr N-PDU but from the FlexRay Interface module point of view.
Fr L-SduId	This is the unique identifier of a Fr-L-SDU within the FlexRay Interface. It is used for referencing L-SDU's routing properties.
Fr N-PDU	This is the PDU of the FlexRay ISO Transport Layer. It contains address information, protocol control information and data (the whole Fr N-SDU or a part of it).
Fr N-SDU	This is the SDU of the FlexRay ISO Transport Layer. In the AUTOSAR architecture, it is a set of data coming from the PDU Router.
Fr N-SduId	Unique N-SDU identifier within the FlexRay ISO Transport Layer. It is used to reference N-SDU's routing properties.
I-PDU	This is the PDU of the upper AUTOSAR Layers modules (e.g. COM, DCM etc.). If data transfer via FlexRay ISO Transport Protocol is configured, I-PDU is similar to an FrIsoTp N-SDU.
PDU	In layered systems, it refers to a data unit that is specified in the protocol of a given layer. This contains user data of that layer (SDU) plus possible protocol control information. Furthermore, the PDU of layer X is the SDU of its lower layer X-1 (i.e. (X)-PDU = (X-1)-SDU).
PduInfoType	This type refers to a structure used to store basic information to process the transmission\reception of a PDU (or a SDU), namely a pointer to its payload in RAM and the corresponding length (in bytes).
Channel	A channel is a resource of the FrIsoTp module to handle communication links to other communication notes from FrIsoTp's point of view (transferring Fr N-PDUs).
Connection	A connection specifies a communication link between different communication notes from FrIsoTp's point of view. A connection defines the sender / receiver relation of communication notes
PCI	Protocol Control Information
e.g.	lat. 'exempli gratia' – engl. for example
i.e.	lat. 'id est' – engl. that is
CanTp	CAN Transport Protocol
LinTp	LIN Transport Protocol
CF	Consecutive Frame Fr N-PDU
COM	AUTOSAR Communications module
DCM	Diagnostic Communication Manager module
DET	Development Error Tracer
FC	Flow Control Fr N-PDU
Fr	FlexRay Driver module
FrIf	FlexRay Interface module
FrIsoTp	FlexRay ISO Transport Protocol Layer
PduR	PDU Router
SF	Start Frame Fr N-PDU
LF	Last Frame Fr N-PDU
TP	Transport Protocol Layer
SDU	In layered systems, this refers to a set of data that is sent by a user of the services of

Acronym:	Description:
	a given layer, and is transmitted to a peer service user, whilst remaining semantically unchanged.
AUTOSAR	Automotive Open System Architecture
SWS	Software Specification
MISRA	Motor Industry Software Reliability Association
ISO	International Standard Organisation
ID	Identifier
HIS	Hersteller Initiative Software
OS	Operating System
MCAL	Microcontroller Abstraction Layer
CPU	Central Processing Unit
ROM	Read Only Memory
RAM	Random Access Memory
STF	Start Frame (please refer to ISO 10681-2)
AF	Acknowledge Frame (please refer to ISO 10681-2)
SN	Sequence Number (please refer to ISO 10681-2)
FrIsoTp_As	Timer Parameter for a sender. Time for transmission of the FlexRay frame (any N_PDU) on the sender side. (please refer to ISO 10681-2)
FrIsoTp_Ar	Timer Parameter for a receiver. Time for transmission of the FlexRay frame (any N_PDU) on the receiver side (please refer to ISO 10681-2)
FrIsoTp_BS	Timer Parameter for a sender. Time until reception of the next FlowControl N_PDU. (please refer to ISO 10681-2)
FrIsoTp_Br	Timer Parameter for a receiver. Time until transmission of the next FlowControl N_PDU. (please refer to ISO 10681-2)
FrIsoTp_Cs	Timer Parameter for a sender. Time until transmission of the next ConsecutiveFrame N_PDU/LastFrame N_PDU. (please refer to ISO 10681-2)
FrIsoTp_Cr	Timer Parameter for a receiver. Time until reception of the next ConsecutiveFrame N_PDU (please refer to ISO 10681-2)

3 Related documentation

3.1 Input documents

[1] List of Basic Software Modules
AUTOSAR_ BasicSoftwareModules.pdf

[2] Layered Software Architecture
AUTOSAR_ LayeredSoftwareArchitecturepdf

[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_FlexRayInterface.pdf

[6] Specification of PDU Router
AUTOSAR_SWS_PDU_Router.pdf

[7] Specification of Communication Stack Types
AUTOSAR_SWS_ComStackTypes.pdf

[8] Specification of Standard Types
AUTOSAR_SWS_StandardTypes.pdf

[9] Specification of Platform Types
AUTOSAR_SWS_PlatformTypes.pdf

[10] Basic Software Module Description Template,
AUTOSAR_BSWMDTemplate.pdf

[11] Specification of ECU Configuration,
AUTOSAR_ ECU_Configuration.pdf

[12] Specification of Diagnostic Event Manager,
AUTOSAR_SWS_DEM.pdf

[13] Specification of Development Error Tracer,
AUTOSAR_SWS_DET.pdf

[14] Specification of FlexRay Transport Layer,
AUTOSAR_SWS_FlexRay_TP.pdf

3.2 Related standards and norms

[15] FlexRay Communications System Protocol Specification Version 2.1

[16] ISO10681-2, Road vehicles – Communication on FlexRay – Part 2: Communication Layer services

[17] HIS MISRA SubSet V2.0
www.automotive-his.de/download/HIS_SubSet_MISRA_C_2.0.pdf

4 Constraints and assumptions

4.1 Limitations

The AUTOSAR architecture defines communication system specific transport protocol layers (FrIsoTp, CanTp, LinTp etc.). Thus, the FlexRay ISO Transport Protocol layer (FrIsoTp) only covers FlexRay ISO transport protocol specifics.

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

4.2 Applicability to car domains

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

4.3 Restriction to ISO 10681-2

The AUTOSAR FrIsoTp module does not support ISO 10681-2 functionalities as listed below:

Functionality	Description
Status monitoring	ISO 10681-2 provides the functionality to monitor the status of an active data transfer. Thus, the ISO-10681-2 API provides the service primitives C_GetStatus.request and C_GetStatus.confirm.
Read Communication Parameter values	ISO 10681-2 provides the functionality to read out current communication parameter values. Thus, the ISO-10681-2 API provides the service primitives C_GetParameters.request and C_GetParameters.confirm.
Receive with unknown message length	ISO 10681-2 provides the functionality to receive the messages with unknown length

Table 1: Limitation of AUTOSAR FrIsoTp vs. ISO 10681-2

5 Dependencies to other modules

This section sets out relations between the FlexRay ISO Transport Protocol (FrIsoTp) and other AUTOSAR basic software modules. It contains brief descriptions of the services, which are required by the FrIsoTp from other modules or which are required by other modules from the FrIsoTp.

5.1 FlexRay ISO Transport Layer interactions

The FrIsoTp's upper interface offers the PduR module global access, to transmit and receive data (Fr N-SDU). FlexRay N-SDU identifiers (Fr N-SDU-ID) achieve this access. FlexRay N- SDU-ID refers to a constant data structure that consists of attributes describing FlexRay N-SDU. The figure below shows the interactions between FrIsoTp, PduR and FrIf modules.

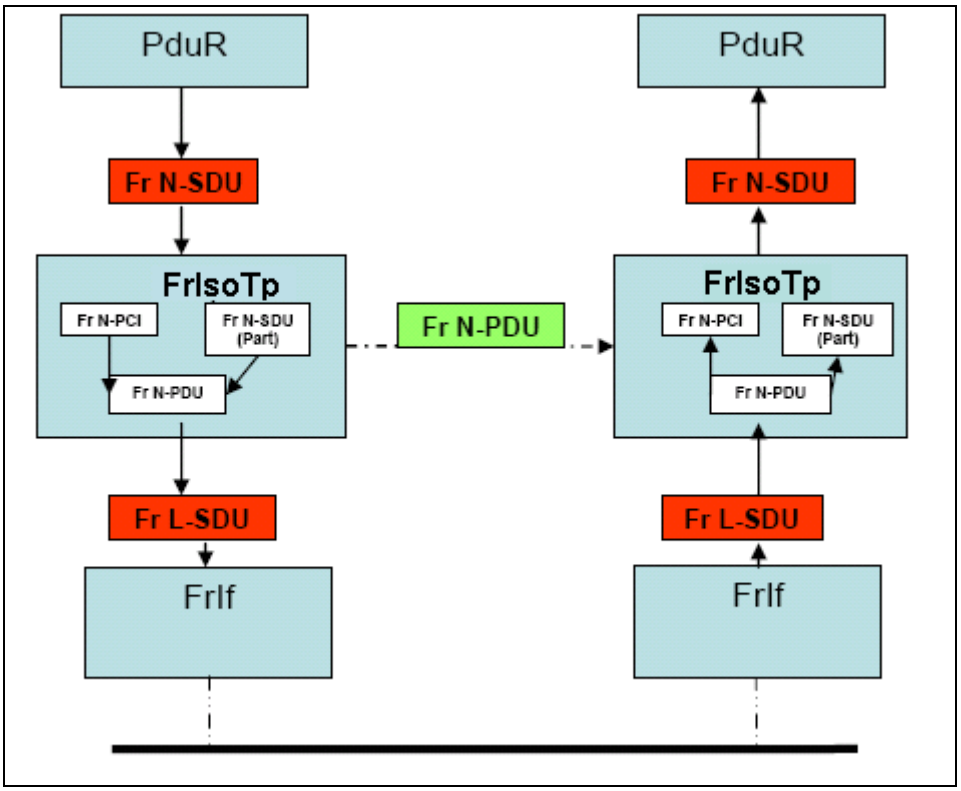


Figure 3: FrIsoTp interactions

5.2 PDU Router

The FrIsoTp module requests different services primitives provided by the PDU Router module. The requested services primitives of the PDU-Router module are listed below. For further details please refer to chapter 8.6 and specification [6]:

-
- ***PduR_FrTpProvideRxBuffer***
  By this API service primitive, the FrIsoTp 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_FrTpRxIndication***
  By this API service primitive, the FrIsoTp indicates the completed (un)successful reception of an FrIsoTp-I-PDU.
- ***PduR_FrTpProvideTxBuffer***
  By this API service primitive, the FrIsoTp asks the actual sender (e.g. DCM) of the message to provide a transmit data. It is not necessary for the transmit data 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 FrIsoTp internal exigencies) will be passed.
- ***PduR_FrTpTxConfirmation***
  By this API service primitive, the FrIsoTp module confirms the (un)successful sending of the complete Fr N-SDU to the corresponding sender module (e.g. COM, DCM etc).

The following services primitives of the FrIsoTp module are called by the PDU-Router:

- ***FrTp_Transmit***
  By this API service primitive, a upper layer initiates a I-PDU data transfer via PDU-Router
- ***FrTp_CancelTransmit***
  By this API service primitive, sending of an Fr N-SDU is cancelled on sender site.
- ***FrTp_ChangeParameter***
  By this API service primitive, some communication parameters of the FrIsoTp module could be changed.
- ***FrTp_CancelReceive***
  By this API service primitive, an ongoing reception could be canceled.

5.3 FlexRay Interface

The following services primitives of the FlexRay Interface (FrIf) module are called by the FrIsoTp:

- ***FrIf_Transmit***
  By this API service primitive, the transfer of an Fr N-PDU is initiated.

The following services primitives of the FrIsoTp module are called by the FlexRay Interface module:

- ***FrTp_RxIndication***
  By this API service primitive, the FlexRay Interface module indicates the reception of an FrIsoTp frame (Fr N-PDU, not to be confused with a FlexRay frame) to the FrIsoTp. The FrIsoTp then processes this frame.
- ***FrTp_TxConfirmation***
  By this API service primitive, the FlexRay Interface module confirms the sending of the frame containing the Fr N-PDU over the FlexRay network.
- ***FrTp_TriggerTransmit***
  By this API service primitive, the FlexRay Interface get access to the Fr N-PDU data.[1]

5.4 ECU State Manager

The following services primitives of the FrIsoTp module are called by the ECU State Manager module (EcuM2859):

- ***FrTp_Init***
  By this API service primitive, the FrIsoTp module is initialized.
- ***FrTp_Shutdown***
  By this API service primitive, all active communication links are closed, resources are freed and the module is stopped.

5.5 Development Error Tracing

The following services primitives of the Development Error Tracing module are called by the FrIsoTp module:

- ***Det_ReportError***
  By this API service primitive, the FrIsoTp module reports development errors.

[1] Depending on the configured buffer access mode.

5.6 File structure

5.6.1 Code file structure

FRISOTP1000: The code-file structure of the FrIsoTp module shall include the file `FrIsoTp.c` containing the source code.

FRISOTP1001: The code-file structure of the FrIsoTp module shall include the file `FrIsoTp_Lcfg.c` containing the link time configurable parameters.

FRISOTP1002: The code-file structure of the FrIsoTp module shall include the file `FrIsoTp_PBcfg.c` containing the post-build time configurable parameters.

5.6.2 Header file structure

FRISOTP195: The header file structure of the FrIsoTp module shall include the file FrIsoTp.h.

FRISOTP1157: FrIsoTp.h shall contain all data exported from the FrIsoTp – API declarations (except callbacks), extern types and global data.

FRISOTP1003: The header file structure of the FrIsoTp module shall include the file FrIsoTp_Cfg.h containing pre-compile time configuration parameters.

FRISOTP1135: The header file structure of the FrIsoTp module shall include the file FrIsoTp_Cbk.h containing all callback function prototypes to be used by the lower layers.

FRISOTP1004: The header file structure of the FrIsoTp module shall include the following files:

- FrIf.h – header file of FrIf,
- MemMap.h – header file for Memory Mapping,
- Det.h – header file of Det,
- SchM_FrIsoTp.h – header file of SchM declarations,
- PduR_FrTp.h – header file of PduR,
- Std_Types.h – header file for standard types
- ComStack_Types.h– header file for ComStack types
- FrIsoTp_Types.h – header file for FrIsoTp specific types
- FrIsoTp_Cfg.h – header file for configuration parameters

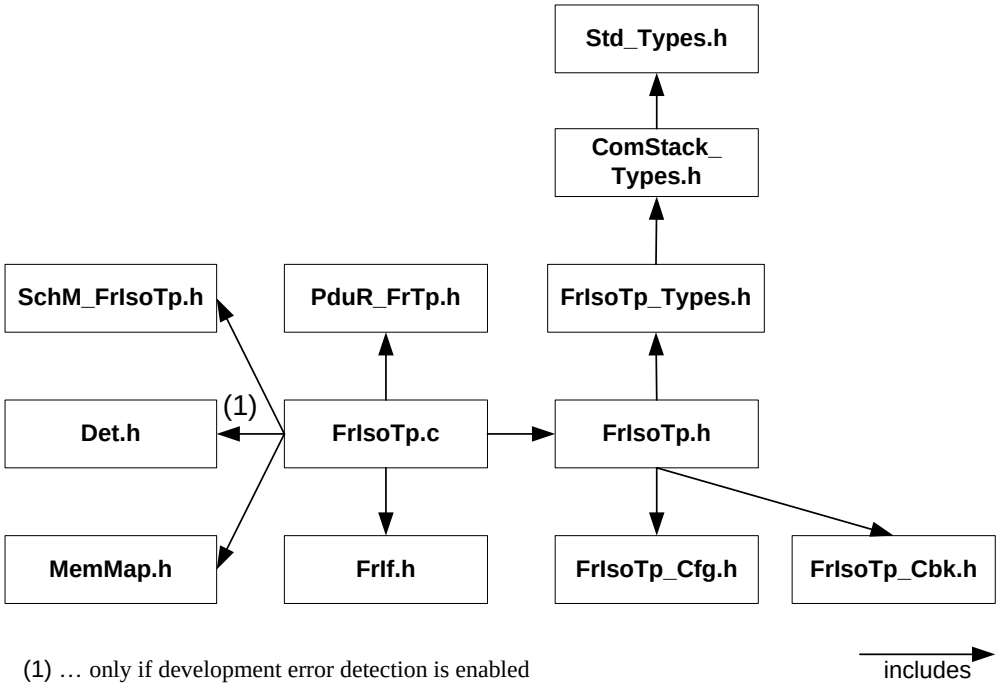


Diagram 1: File Structure

The structure as depict in Diagram 1 allows the separation between platform, compiler and implementation specific definitions and declarations from general definitions as well as the separation of source code and configuration.

5.6.3 Module Files Consistency

FRISOTP200: Each code (*.c) file of the FrIsoTp module shall provide data of version identification as defined in chapter 10.3.

FRISOTP1158: Each header (*.h) file of the FrIsoTp module shall provide data of version identification as defined in chapter 10.3.

FRISOTP201: The FrIsoTp module shall perform a pre-processor-check to analyze whether the included source and header files have identical version numbers.

5.6.4 Design rules

FRISOTP208: The code of the FrIsoTp module (as long as it is written in C) shall conform to the HIS subset of the MISRA C Standard [17].

FRISOTP209: The source code of the FrIsoTp module shall be neither compiler (tool) nor platform (processor) dependent.[2]

[2] No compiler specific keywords shall be used.

FRISOTP210: The code of the FrIsoTp module (as long as it is written in C) shall indicate all global data with read-only purposes by explicitly assigning the `const` keyword.

FRISOTP211: The code of the FrIsoTp module may provide macros instead of functions where source code is used and runtime is critical.

FRISOTP212: All data/variables that shall be debugged and traced by AUTOSAR Debugging have to be defined as global variables.

FRISOTP1159: Type definitions of global variables, which shall be debugged and traced by AUTOSAR, shall be accessible by the standard module header file FrIsoTp.h.[3]

FRISOTP1129: The FlexRay ISO Transport Layer module architecture shall support configuration modification by a dedicated update process (e.g. flash reprogramming)

[3] This allows the debugging toolchain to calculate the size of elements by C-"sizeof" and to (optionally) decode the structure elements.

6 Requirements traceability

6.1 General Requirements on Basic Software Modules

Requirement		*Satisfied by*
BSW003	Version identification	
BSW00300	Module naming convention	Chapter 8
BSW00301	Limit imported information	n/a
BSW00302	Limit exported information	n/a
BSW00304	AUTOSAR integer data types	Chapter 8
BSW00305	Self-defined data types naming convention	**FRISOTP1133**
BSW00306	Avoid direct use of compiler and platform specific keywords	n/a
BSW00307	Global variables naming convention	n/a
BSW00308	Definition of global data	FRISOTP212
BSW00309	Global data with read-only constraint	FRISOTP210
BSW00310	API naming convention	Chapter 8
BSW00312	Shared code shall be reentrant	n/a
BSW00314	Separation of interrupt frames and service routines	n/a
BSW00318	Format of module version numbers	**FRISOTP001_PI**
BSW00321	Enumeration of module version numbers	**FRISOTP001_PI**
BSW00323	API parameter checking	n/a
BSW00325	Runtime of interrupt service routines	n/a
BSW00326	Transition from ISRs to OS tasks	n/a
BSW00327	Error values naming convention	Chapter 7.7.3
BSW00328	Avoid duplication of code	n/a
BSW00329	Avoidance of generic interfaces	n/a
BSW00330	Usage of macros / inline functions instead of functions	n/a
BSW00331	Separation of error and status values	n/a
BSW00333	Documentation of callback function context	n/a
BSW00334	Provision of XML file	n/a
BSW00335	Status values naming convention	n/a
BSW00336	Shutdown interface	Chapter 7.4
BSW00337	Classification of errors	Chapter 7.7.3
BSW00338	Detection and Reporting of development errors	Chapter 7.7.3
BSW00339	Reporting of production relevant error status	Chapter 7.7.3
BSW00341	Microcontroller compatibility documentation	n/a
BSW00342	Usage of source code and object code	n/a
BSW00343	Specification and configuration of time	n/a
BSW00344	Reference to link-time configuration	**FRISOTP1001**
BSW00345	Pre-compile-time configuration	n/a
BSW00346	Basic set of module files	Chapter 5.6
BSW00347	Naming separation of different instances of BSW drivers	n/a
BSW00348	Standard type header	Chapter 5.6
BSW00350	Development error detection keyword	n/a
BSW00353	Platform specific type header	Chapter 5.6
BSW00355	Do not redefine AUTOSAR integer data types	Chapter 8
BSW00357	Standard API return type	Chapter 8.3
BSW00358	Return type of init() functions	n/a
BSW00359	Return type of callback functions	Chapter 8.2.1
BSW00360	Parameters of callback functions	Chapter 8.2.1
BSW00361	Compiler specific language extension header	Chapter 5.6
BSW00369	Do not return development error codes via API	Chapter 7.7.3
BSW00370	Separation of callback interface from API	n/a
BSW00371	Do not pass function pointers via API	n/a
BSW00373	Main processing function naming convention	n/a

BSW00374	Module vendor identification	**FRISOTP001_PI**
BSW00375	Notification of wake-up reason	n/a
BSW00376	Return type and parameters of main processing functions	n/a
BSW00377	Module specific API return types	n/a
BSW00378	AUTOSAR boolean type	Chapter 8
BSW00379	Module identification	**FRISOTP001_PI**
BSW00380	Separate C-Files for configuration parameters	**FRISOTP1000**, **FRISOTP1001**, **FRISOTP1002**
BSW00381	Separate configuration header file for pre-compile time parameters	**FRISOTP1000**, **FRISOTP1001**, **FRISOTP1002**
BSW00383	List dependencies of configuration files	Chapter 5
BSW00384	List dependencies to other modules	Chapter 5
BSW00385	List possible error notifications	Chapter 8.2.1
BSW00386	Configuration for detecting an error	n/a
BSW00387	Specify the configuration class of callback function	n/a
BSW00388	Introduce containers	Chapter 10.2
BSW00389	Containers shall have names	Chapter 10.2
BSW00390	Parameter content shall be unique within the module	Chapter 10.2
BSW00391	Parameter shall have unique names	Chapter 10.2
BSW00392	Parameters shall have a type	Chapter 10.2
BSW00393	Parameters shall have a range	Chapter 10.2
BSW00394	Specify the scope of the parameters	Chapter 10.2
BSW00395	List the required parameters (per parameter)	Chapter 10.2
BSW00396	Configuration classes	Chapter 10.2
BSW00397	Pre-compile-time parameters	Chapter 10.2
BSW00398	Link-time parameters	Chapter 10.2
BSW00399	Loadable Post-build time parameters	Chapter 10.2
BSW004	Version check	**FRISOTP200**, **FRISOTP201**
BSW00400	Selectable Post-build time parameters	Chapter 10.2
BSW00401	Documentation of multiple instances of configuration parameters	n/a
BSW00402	Published information	Chapter 10.3
BSW00404	Reference to post build time configuration	**FRISOTP1002**
BSW00405	Reference to multiple configuration sets	n/a
BSW00406	Check module initialization	Chapter 7.4
BSW00407	Function to read out published parameters	**FRISOTP215**
BSW00408	Configuration parameter naming convention	Chapter 10.2
BSW00409	Header files for production code error IDs	n/a
BSW00410	Compiler switches shall have defined values	n/a
BSW00411	Get version info keyword	**FRISOTP215**
BSW00412	Separate H-File for configuration parameters	**FRISOTP1003**
BSW00413	Accessing instances of BSW modules	n/a
BSW00414	Parameter of init function	n/a
BSW00415	User dependent include files	n/a
BSW00416	Sequence of Initialization	Chapter 7.4
BSW00417	Reporting of Error Events by Non-Basic Software	n/a
BSW00419	Separate C-Files for pre-compile time configuration parameters	**FRISOTP1000**, **FRISOTP1001**, **FRISOTP1002**
BSW00422	Pre-Debouncing of production relevant error status	n/a
BSW00423	Usage of SW-C template to describe BSW modules with AUTOSAR Interfaces	n/a

BSW00424	BSW main processing function task allocation	n/a
BSW00425	Trigger conditions for schedulable objects	n/a
BSW00426	Exclusive areas in BSW modules	n/a
BSW00427	ISR description for BSW modules	n/a
BSW00428	Execution order dependencies of main processing functions	n/a
BSW00429	Restricted BSW OS functionality access	n/a
BSW00431	The BSW Scheduler module implements task bodies	n/a
BSW00432	Modules should have separate main processing functions for read/receive and write/transmit data path	n/a
BSW00433	Calling of main processing functions	n/a
BSW00434	The Schedule Module shall provide an API for exclusive areas	n/a
BSW00435	Header File Structure for the Basic Software Scheduler	Chapter 5.6.2
BSW00436	Module Header File Structure for the Memory Mapping	Chapter 5.6.2
BSW005	No hard coded horizontal interfaces within MCAL	n/a
BSW006	Platform independency	n/a
BSW007	HIS MISRA C	**FRISOTP208**
BSW009	Module User Documentation	n/a
BSW010	Memory resource documentation	n/a
BSW101	Initialization interface	**FRISOTP147**
BSW158	Separation of configuration from implementation	Chapter 5.6.1
BSW159	Tool-based configuration	n/a
BSW160	Human-readable configuration data	n/a
BSW161	Microcontroller abstraction	n/a
BSW162	ECU layout abstraction	n/a
BSW164	Implementation of interrupt service routines	n/a
BSW167	Static configuration checking	n/a
BSW168	Diagnostic Interface of SW components	n/a
BSW170	Data for reconfiguration of AUTOSAR SW-components	n/a
BSW171	Configurability of optional functionality	Chapter 10.4
BSW172	Compatibility and documentation of scheduling strategy	n/a

6.2 Requirements on FlexRay

Requirement		***Satisfied by***
BSW050xx	Compliance with ISO 10681-2	**FRISOTP1005**, **FRISOTP1006**
BSW05074	Location of the FlexRay Transport Layer software module.	Chapter 1
BSW05075	Independence of the Network Configuration.	Chapter 1
BSW05076	Support of at least 32 logical FlexRay Transport Layer Channels being used concurrently.	FRISOTP004_Conf
BSW05077	Identification of each N-SDU with a unique identifier.	**FRISOTP1018**
BSW05079	Individual properties of each N-SDU.	Chapter 10.2
BSW05082	Support acknowledgement without retry.	n/a
BSW05083	Support acknowledgement with retry.	**FRISOTP1005**, **FRISOTP1006**
BSW05088	Interface for initialization.	**FRISOTP1034**, **FRISOTP1035**
BSW05089	The FlexRay Transport Layer services shall not be operational before initializing the module.	**FRISOTP1037**
BSW050xx	Support of the Change Parameter Service according to ISO 10681-2.	Chapter 7.5.7
BSW05093	Provide a transmit cancellation service to the sender and the receiver.	**FRISOTP1005**, **FRISOTP1006** Chapter 7.5.6
BSW05095	Dynamic control of used bandwidth.	**FRISOTP1005**,

		FRISOTP1006
BSW05123	Modification of configuration data by a flashing process	**FRISOTP1129**

7 Functional specification

This section provides a description of the FlexRay ISO Transport Protocol Layer functionality. It explains the services provided to the upper and lower layers and the internal behavior of the FrIsoTp Layer module.

The main purpose of the FlexRay ISO Transport Protocol (FrIsoTp) Layer is transferring messages (I-PDUs) that may or may not fit in a single FlexRay frame (L-PDU). The FrIsoTp Layer module provides services for segmentation and reassembly of upper-layer messages (Fr N-SDUs). Hence FrIsoTp module offers services for segmentation, transmission with flow control, and reassembly of messages.

While reading this document, it is necessary to bear in mind, that the Transport Protocol functionality (e.g. frame assembly, frame handling, error handling etc.) is according to ISO10681-2, Road vehicles – Communication on FlexRay – Part 2: Communication Layer services [16].

FRISOTP1005: If no explicit recommendation or requirement is defined, the FrIsoTp module shall follow the specification ISO10681-2, Road vehicles – Communication on FlexRay – Part 2: Communication Layer services [16].

Transport protocol facilities will be used to transport AUTOSAR I-PDUs from different source modules (e.g. COM, DCM etc.). Therefore, the FrIsoTp module is able to deal with multiple connections simultaneously (i.e. multiple segmentation sessions in parallel).

The maximum number of simultaneous active connections is statically configured. This configuration has an important impact on complexity and resource consumption (CPU, ROM and RAM) of the code generated, because resources (e.g. Rx and Tx state machines, variables used to work on N-PCI data and so on) have to be reserved for each simultaneous access.

7.1 FrIsoTp usage scenarios

As depict in Figure 4, the FrIsoTp module is usable within single Electronic Control Units (ECUs) as well as in Gateways. In both cases FrIsoTp modules are responsible to handle communication via FlexRay but for gateway purpose some additional requirements have to be taken into account.

Note: Each time a special usage scenario has to be taken into account, a foot node is given within the document.

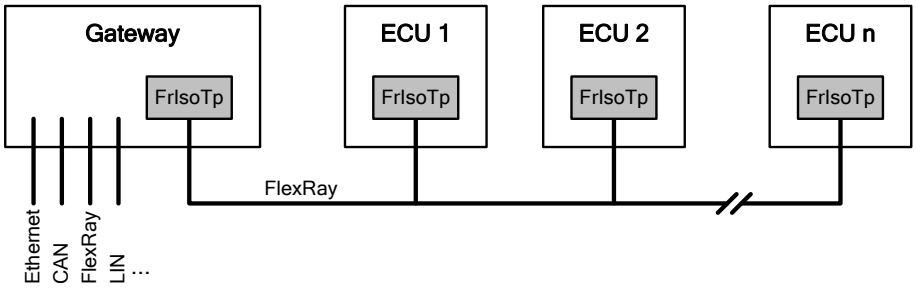


Figure 4: FrIsoTp usage scenarios

7.2 FrIsoTp behavior according to ISO10681-2

This chapter gives a small overview about the Transport Protocol behaviour and data transmission scenarios according to ISO 10681-2. This chapter is only for a better understanding of the software solution to fullfill ISO 10681-2 and specifies no additional requirement.

7.2.1 Protocol Data Unit (PDU)

FRISOTP1006: The FrIsoTp module shall support the Fr N-PDU formats as defined in [16] ISO10681-2, Road vehicles – Communication on FlexRay – Part 2: Communication Layer services

7.2.2 Frame Sequence charts

This chapter describes the data transfer modes based on Transport Protocol Layer frame (N-PDU) sequences according to ISO10681-2. This is only for a better understanding of the FrIsoTp internal work and shall not be an additional specification. For a final implementation only the figures in [16] ISO10681-2, Road vehicles – Communication on FlexRay – Part 2: Communication Layer services are relevant.

7.2.2.1 Unsegmented unacknowledged data transfer with known message length

FRISOTP1007: According to ISO 10681-2 [16] the FrIsoTp module shall support an unsegmented unacknowledged data transfer with known message length as depict in Figure 5.

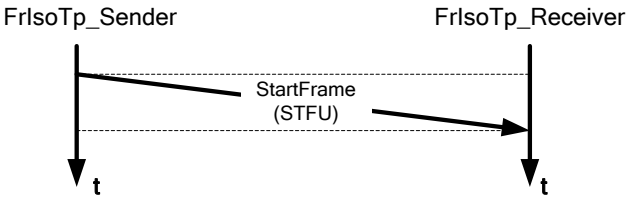


Figure 5: Frame sequence of an unsegmented unacknowledged data transfer with known message length

7.2.2.2 Unsegmented acknowledged data transfer with known message length

FRISOTP1008: According to ISO 10681-2 [16] the FrIsoTp module supports an unsegmented acknowledged data transfer with known message length as depict in Figure 6.

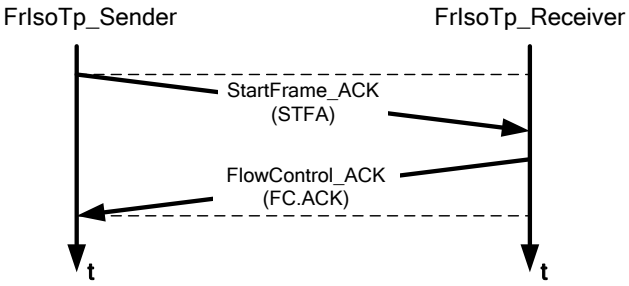


Figure 6: Frame sequence of an unsegmented acknowledged data transfer with known message length

7.2.2.3 Segmented unacknowledged data transfer with known message length

FRISOTP1009: According to ISO 10681-2 [16] the FrIsoTp module shall support a segmented unacknowledged data transfer with known message length as depict in Figure 7.

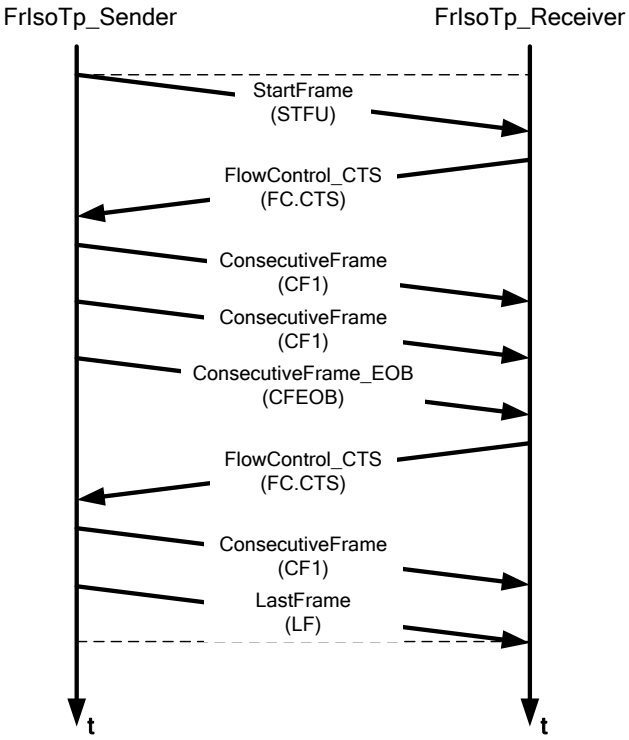


Figure 7: Frame sequence a segmented unacknowledged data transfer with known message length

7.2.2.4 Segmented acknowledged data transfer with known message length

FRISOTP1010: According to ISO 10681-2 [16] the FrIsoTp module shall support a segmented acknowledged data transfer with known message length as depict in Figure 8.

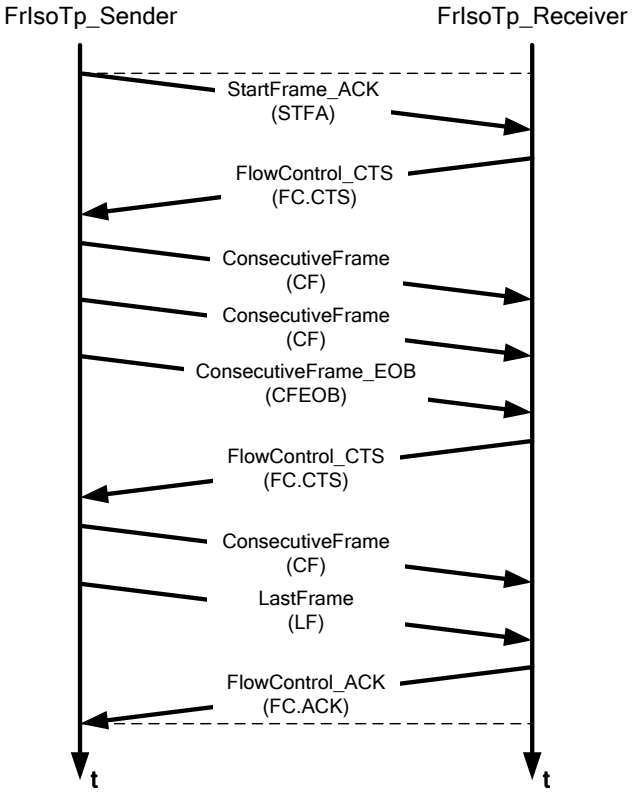


Figure 8: Frame sequence of a segmented acknowledged data transfer with known message length

7.2.2.5 Segmented unacknowledged data transfer with unknown message length

FRISOTP1011: According to ISO 10681-2 [16] the FrIsoTp module shall support a segmented unacknowledged data transfer with unknown message length as depict in Figure 9.

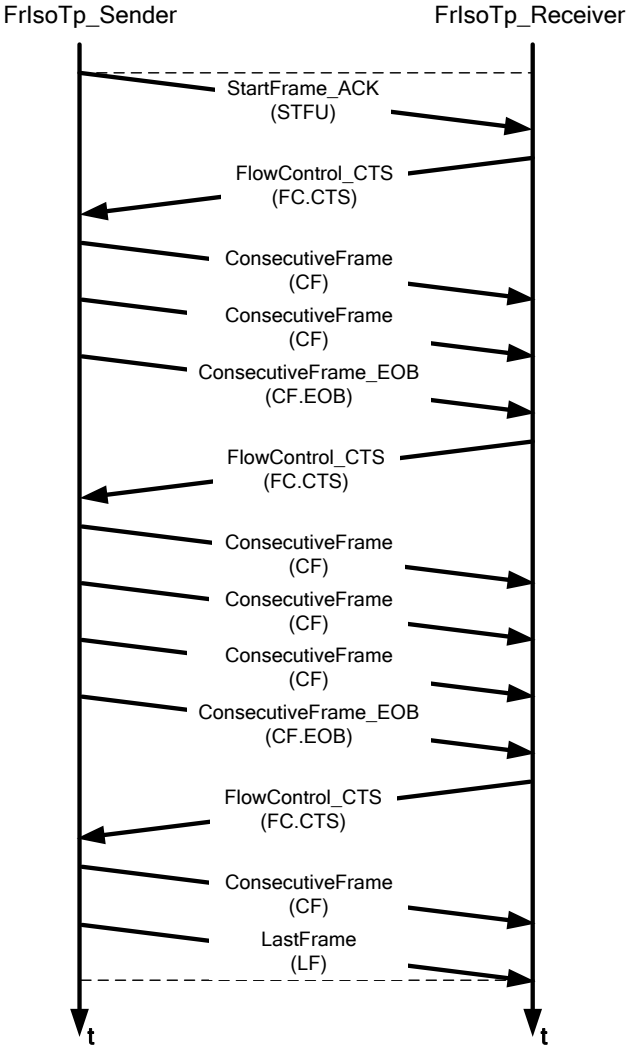


Figure 9: Frame sequence of a segmented unacknowledged data transfer with unknown message length

7.2.2.6 Segmented acknowledged data transfer with unknown message length

FRISOTP1012: According to ISO 10681-2 [16] the FrIsoTp module shall support a segmented acknowledged data transfer with unknown message length as depict in Figure 10.

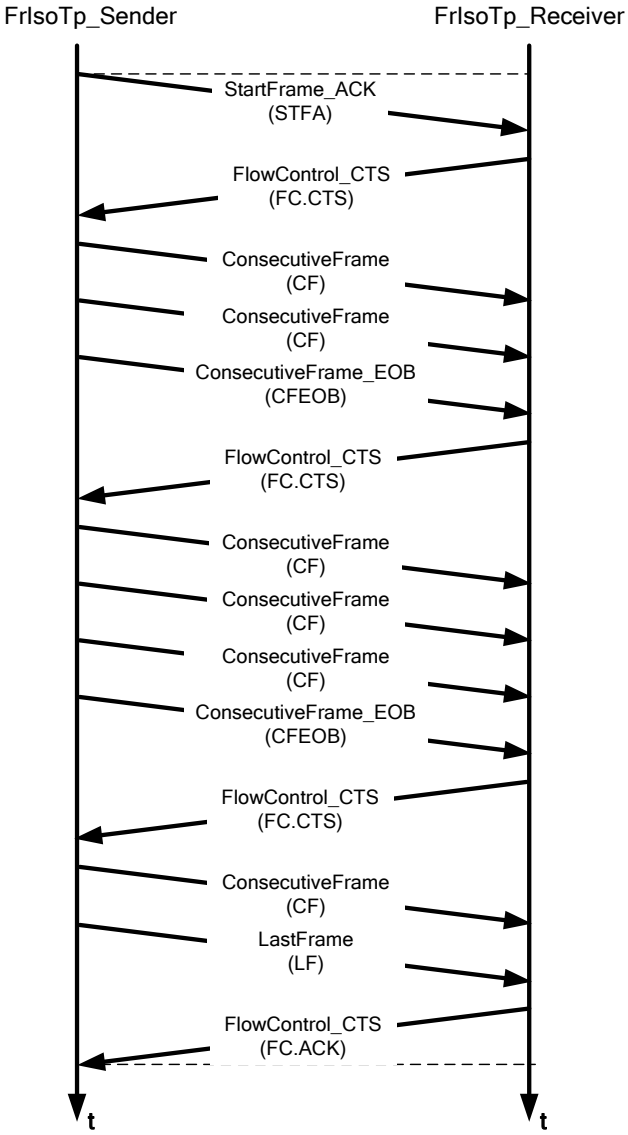


Figure 10: Frame sequence of a segmented acknowledged data transfer with unknown message length

7.2.3 Limitation to ISO10681-2

The limitations to ISO 10681-2 are described in chapter 4.3 - Table 1.

7.3 Internal Module behavior specification

This chapter specifies the internal behaviour of the FlexRay ISO Transport Layer module to fullfill the protocol behaviour according to ISO 10681-2 [13].

7.3.1 Overview

Figure 11 depicts an abstract overview of the FrIsoTp layer module architecture.

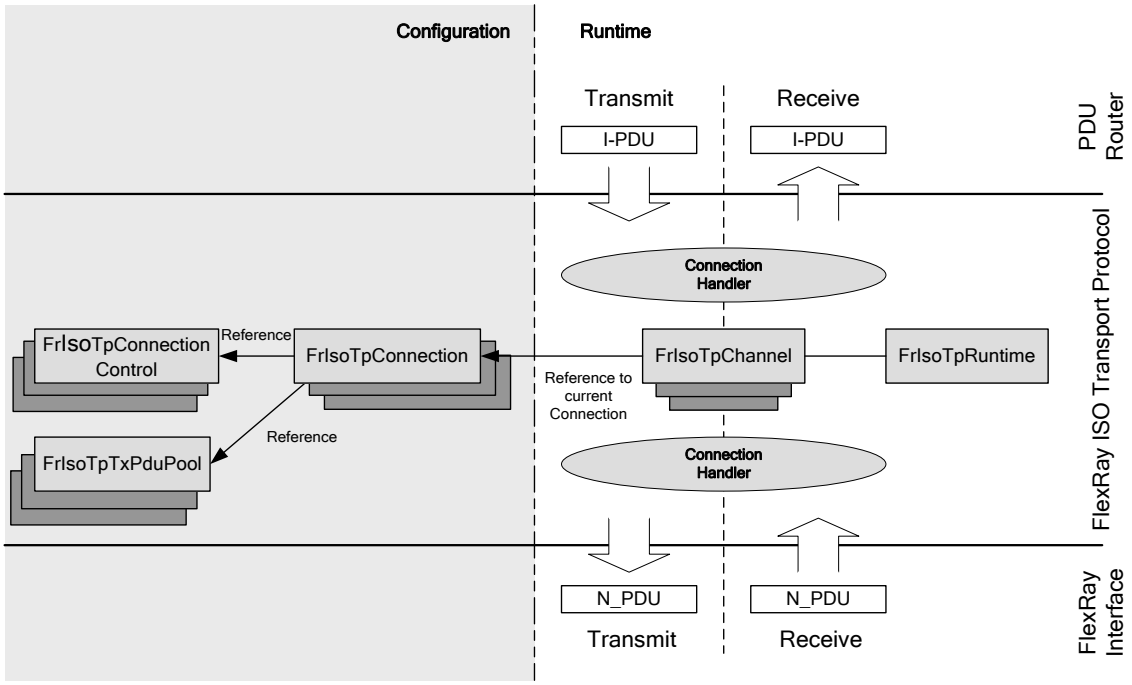


Figure 11: FlexRay ISO Transport Module overview

Figure 11 depicts a division between configuration parts and runtime parts. After the module is initialized it is able to transmit I-PDUs from an upper layer (PDUR) or receive N-PDUs from a lower layer (FrIf). Below there is a short describes of the different parts being involved in FrIsoTp layer handling procedure.

Term	Description
FrIsoTpConnection	A connection is a configuration parameter set which includes all parameters to identify a connection link between different communication nodes uniquely. A connection has a fixed assignment between a sender node (representing by a source address), one or more receiver node(s) (representing by a target address), the upper Layer I-PDU source (representing by an I-PDU-ID). Additionally a connection has a reference to a set of N-PDUs (PDU-Pool) which are defined for sending data via FrIsoTp. A reference to connection specific parameters (e.g. timings and timeouts etc.) is defined too.
FrIsoTpConnection Control	A connection configuration is a parameter set which includes configuration specific parameter (e.g. timings,

	timeouts, default parameters etc.). It is referenced by a connection.
FrIsoTpTxPduPool	A Tx-N-PduPool is a set of N-PDUs which are defined for FrIsoTp sending purpose.
FrIsoTpChannel	A channel is a runtime resource of the FrIsoTp module which implements all communication control mechanisms (e.g. state machine etc.) to handle a communication link via FlexRay. A channel could be allocated by the connection handler to process a required connection. Therefore a channel has a reference to the connection which is currently handled by this channel. If a data transfer has been finished the assignment between the channel and the connection is cleared and the channel could be reallocated by another connection.
FrIsoTpRuntime	FrIsoTpRuntime is a set of runtime parameters which is necessary to control active connections. (please refer to chapter 7.3.3.1)
Connection Handler	The connection handler is an abstract part of the FrIsoTp module and is responsible for the (re-)allocation of channels and the (re-)assignment of channels and connections.

If an upper layer module (e.g. COM, DCM etc) wants to transmit data (I-PDU) the PduR module executes the corresponding FrIsoTp layer module API call. The connection handler evaluates the I-PDU-ID (equal to N-SDU-ID from FrIsoTp's point of view) for the corresponding connection. The connection handler allocates a free channel and set channel's connection reference to the selected connection. The channel could now initialize with the connection control parameter set which is access able via the references. The channel process the communication until all data have been transmitted. After the last N-PDU was send the connection handler will free the channel and also the reference to the connection is reset.

If an N-PDU is received via FlexRay the FrIf executes the corresponding FrIsoTp API call. The connection handler evaluates the target address and the source address of the N-PDU which is part of the Protocol Control Information (PCI). The connection handler search for the corresponding connection, allocates a channel, set the reference to the selected connection and initialize them. Until the last N-PDU was received the connection handler reallocates the channel, skips the reference to the selected connection and delivers the I-PDU to the addressed upper layer by calling the corresponding PDUR-module API.

7.3.2 Configuration data

7.3.2.1 FrIsoTpConnection

A connection identifies the sender and the receiver(s) of this particular communication link.
An FrIsoTp connection link is defined by
a) a target address of the receiver node(s) and

b) a source address of the sender node.

For the internal handling of different PDUs across the FlexRay communication stack the I-PDU-ID (N-SDU-ID) identifies the data link to upper layer's sender or receiver modules (see Figure 2).
Additionally a connection has a reference to a set of N-PDUs (`FrIsoTpTxPduPool`) which are defined for sending data via this particular connection. A reference to connection specific parameters, e.g. timings and timeouts etc is defined too (FrIsoTpConnectionConfiguration).

FRISOTP1013: An `FrIsoTpConnection` container shall implement all parameters as defined in chapter 10.2.

FRISOTP1014: For each connection link the FrIsoTp module shall handle, a new instance of `FrIsoTpConnection` container shall be created.

FRISOTP1015: The FrIsoTp module shall support a post build time configurable number of connections[4].

FRISOTP1017: Each `FrIsoTpConnection` shall have a module wide unique `RemoteAddress / LocalAddress` pair `(see` section 10.2`)`.[5]

FRISOTP1018: Each `FrIsoTpConnection` shall have a module wide unique `FRISOTP_SDUID` (N-SDU ID) `(see` section 10.2`)`.

7.3.2.2 FrIsoTpTxPduPool

The `FrIsoTpTxPduPool` contains a list of N-PDUs configured for sending FrIsoTp N-PDUs. An `FrIsoTpTxPduPool` could be referenced by different `FrIsoTpConnections` but each `FrIsoTpConnection` has exactly one reference to one `FrIsoTpTxPduPool`. (see also Figure 17). The `FrIsoTpTxPduPools` are necessary to support dynamic bandwidth assignment for connections.

Chapter 7.5.5 describes the dynamic bandwidth assignment in detail. At this position in specification only the term `FrIsoTpTxPduPool` shall be introduced and some basic requirements to `FrIsoTpTxPduPools` are specified.

FRISOTP1019: An `FrIsoTpTxPduPool` container shall implement all parameters as defined in chapter 10.2.

FRISOTP1020: A single `FrIsoTpTxPduPool` can be referenced by different `FrIsoTpConnections`.

[4] Post-build time configurable number of connections is required e.g. for gateways. If new connections are defined during vehicle lifecycle only the connection's parameter set has to be updated.
[5] The AUTOSAR local address and remote address is mapped to the ISO 10681-2 source address and target address.

Note: Configuration of PDU Pools to limit bandwidth to an ECU is described in chapter 10.4.5.

7.3.2.3 FrIsoTpConnectionControl

An `FrIsoTpConnectionControl` container contains all static (not runtime) parameters, which are necessary to control a connection e.g. initial timer values, timeout control values etc. Each `FrIsoTpConnection` has an exclusive link to an `FrIsoTpConnectionControl` container. `FrIsoTpConnections` with equal control parameters can reference the same `FrIsoTpConnectionControl`[6].

FRISOTP1021: An `FrIsoTpConnectionControl` Container shall implement all parameters as defined in chapter 10.2.

FRISOTP1022: An `FrIsoTpConnectionControl` container can be referenced by different `FrIsoTpConnections`.

7.3.3 Runtime data

As depict in Figure 11 also some runtime information are required. All runtime information are encapsulated in containers. This chapter defines all the runtime containers with the corresponding variables in that scope as it necessary to understand FrIsoTp's work.
It's recommended to place all runtime data required for implementation into the global `FrIsoTpRuntime` container too.

7.3.3.1 FrIsoTpRuntime

Module Name	**FrIsoTpRuntime**
Module Description	This container contains the runtime parameters / variables which are necessary to handle FlexRay Transport Protocol communication according to ISO 10681-2.

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
FrIsoTp_Channel	1..*	FrIsoTp Channel: This container contains the runtime parameters / variables for an FrIsoTpChannel.
FrIsoTp_ConCtrlRuntime	1..*	FrIsoTp Connection Control Runtime: This container contains the runtime parameters / variables to handle an FrIsoTpConnection.
FrIsoTp_PduPoolRuntime	1..*	FrIsoTp Pdu Pool Runtime: This container contains the runtime parameters / variables to handle an FrIsoTpPduPool.

[6] Use case: Reducing configuration control container instances

7.3.3.2 FrIsoTpChannel

As described above a channel is a runtime resource of the FrIsoTp. A channel could be allocated to handle a connection. This chapter describes the relevant information of a channel without implementation specific information (e.g. types etc).

Name	FrIsoTpChannel
Description	This container contains the parameters and variables of a FlexRay channel
Container parameters and variables	

Information	Description
FrIsoTpChannelNumber	Number of that channel
FrIsoTpTxChannelState	Current state of the Tx channel (idle = 0 or busy = 1)
FrIsoTpTxConState	FrIsoTp Tx Connection State: This parameter implements the current state of the Tx connection (Tx communication state machine according to ISO 10681-2 protocol handling).
FrIsoTpTxConRef	FrIsoTp Tx Connection Reference: This is the reference (pointer to connection) to the current Tx *FrIsoTpConnection*, the channel is currently processing.
FrIsoTpTxConTxPduPendingCounter	FrIsoTp Tx Connection Tx Pdu Pending Counter: This counter counts the number of currently transmitted but not confirmed Fr N-PDUs of the active TxConnection (e.g. SF, CF, LF) This counter shall be incremented each time an FrIsoTp Tx N-PDU is send by the FrIsoTp. This counter shall be decremented each time an transmitted FrIsoTp Tx N-PDU is confirmed by the FrIf.
FrIsoTpTxConTxPduPoolRuntimeRef	FrIsoTp TxConnection Tx PDU Pool Runtime Reference: This is the reference (pointer to FrIsoTp_TxPduPoolRuntime) to the runtime container of the corresponding PDU-Pool. Note: The runtime container of the PDU Pool controls the TxConfirmation signals.
FrIsoTpTxConConfigRuntimeRef	FrIsoTp Tx Connection Configuration Runtime Reference: This is the reference (pointer to FrIsoTp_ConConfigRuntime) to the runtime container of the corresponding Tx connection configuration. Note: The runtime container of the Tx

	connection configuration controls the connection parameters, which are changeable during runtime.
FrIsoTpRxChannelState	Current state of the Rx channel (idle or busy)
FrIsoTpRxConState	FrIsoTp Rx Connection State: This parameter implements the current state of the Rx connection (Rx communication state machine according to ISO 10681-2 protocol handling).
FrIsoTpRxConRef	FrIsoTp Rx Connection Reference: This is the reference (pointer to connection) to the current Rx *FrIsoTpConnection*, the channel is currently processing.
FrIsoTpRxConTxPduPendingCounter	FrIsoTp Rx Connection Tx Pdu Pending Counter: This counter counts the number of currently transmitted but not confirmed Fr N-PDUs of the active RxConnection (FlowControl). This counter shall be incremented each time an FrIsoTp Tx N-PDU is send by the FrIsoTp. This counter shall be decremented each time a transmitted FrIsoTp Tx N-PDU is confirmed by the FrIf. Therefore that FrIsoTp Rx Connection Tx Pdu Pending Counter toggles only between 0 and 1.
FrIsoTpRxConTxPduPoolRuntimeRef	FrIsoTp Rx Connection Tx PDU Pool Runtime Reference: This is the reference (pointer to FrIsoTpTxPduPoolRuntime) to the runtime container of the corresponding PDU-Pool. Note: The runtime container of the PDU Pool controls the TxConfirmation signals. This reference is required for the transmission control of FlowControl N-PDU during an ongoing reception on that channel. Note: For a full duplex channel configuration (see chapter 7.3.3.2.1) the FrIsoTpTxConTxPduPoolRuntimeRef and the FrIsoTpRxConTxPduPoolRuntimeRef are equal.
FrIsoTpRxConConfigRuntimeRef	FrIsoTp Rx Connection Configuration Runtime Reference: This is the reference (pointer to FrIsoTpConConfigRuntime) to the runtime container of the corresponding Rx

	connection configuration. Note: The runtime container of the Rx connection configuration controls the connection parameters, which are changeable during runtime.
No included containers	

FRISOTP228: The FrIsoTp module shall support concurrently work of multiple `FrIsoTpChannels`[7].

FRISOTP088: The exact number of provided channels shall be configurable by the parameter FrIsoTpChanNum (see section 10.2)

FRISOTP1025: The runtime variable *FrIsoTpRxChannelState* shall be switched from "`idle`" state to "`busy`" state if the channel is allocated for an Rx connection.

FRISOTP1026: The runtime variable *FrIsoTpRxChannelState* shall be switched from "`busy`" state to "`idle`" state if the channel is free after an Rx connection is closed.

FRISOTP1117: The runtime variable *FrIsoTpTxChannelState* shall be switched from "`idle`" state to "`busy`" state if the channel is allocated for an Tx connection.

FRISOTP1118: The runtime variable *FrIsoTpTxChannelState* shall be switched from "`busy`" state to "`idle`" state if the channel is free after an Tx connection is closed.

Note: The error handling for the case if no FrIsoTpChannel resource is available (*FrIsoTpTxChannelState* ≠ idle) for data transmission is specified in **FRISOTP1041**.

7.3.3.2.1 Full Duplex and Half Duplex

Normally a Full Duplex channel supports concurrent transmission and reception of Fr N-PDUs at the same time for the same[8] connection. Figure 12 depicts a Full Duplex implementation.

7 The number of channels represents the number of connections, which could be handled concurrently for the same direction. Therefore it is an indication of the performance of the FrIsoTp. On the other hand a Gateway requires more channels than normal ECUs because a gateway handles more concurrent connections. Hence the number of supported channels shall be configurable.

8 Theoretically it is possible that two ECUs transferring data to each other at the same time. That means that each ECU is sender and receiver concurrently. If both ECUs have only one Remote Address and one Local Address the FrIsoTp shall evaluate the PCI to distinguish whether a PDU for Rx-Direction (CF or LF) or a PDU for Tx-direction (FC) was received. This is a full duplex mechanism.

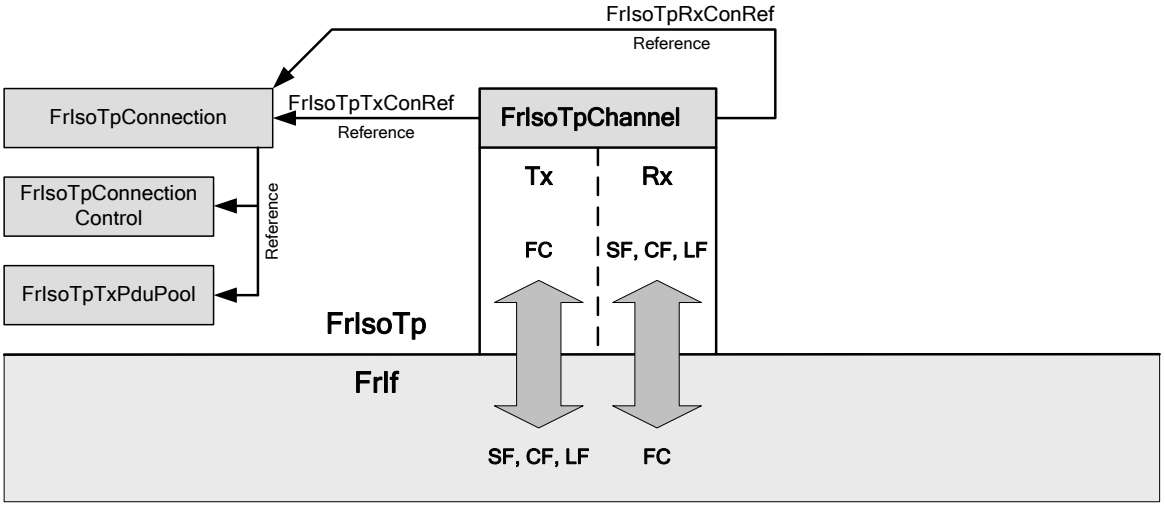


Figure 12: Full Duplex Overview

On the other hand a Half Duplex channel supports only a data transfer for one direction. The fact that an Rx transmission has also to send a FlowControl or a Tx transmission has to receive a FlowControl is not similar to a full duplex connection. Figure 13 depicts a half duplex FrIsoTp_Channel, where either a Tx or a Rx Connection is processed.

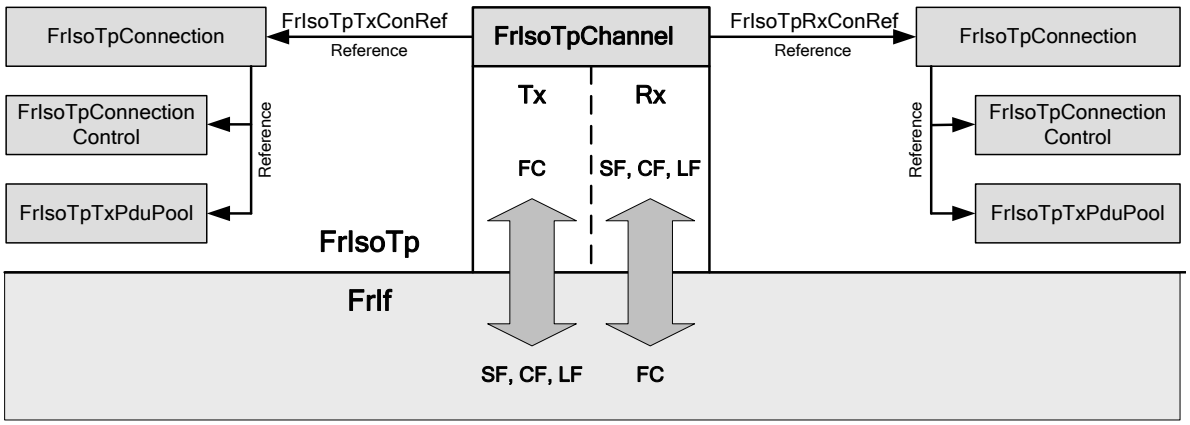


Figure 13: Half Duplex Overview

The final functionality of FrIsoTpChannels depends on implementation and therefore it is not specified in this document.

7.3.3.3 FrIsoTpConnectionControlRuntime

This chapter describes the relevant information of Connection Control without implementation specific information (e.g. types etc).

Name	FrIsoTpConCtrlRuntime
Description	FrIsoTp Connection Control Runtime: This container contains the ConnectionControl runtime data.
Container parameters and variables	

Information	Description
FrIsoTpSCexpRuntime	FRISOTP_SEPARATION_CYCLE_EXPONENT Runtime value of FrIsoTpSCexp parameter. This parameter could be changed via API-Call “FrTp_ChangeParameter” and differs than from the configured default value.
FrIsoTpMaxNbrOfNPduPerCycleRuntime	FRISOTP_MAX_NUMBER_OF_NPDU_PER_CYCLE Runtime value of FrIsoTpMaxNbrOfNPduPerCycle parameter. This parameter could be changed via API-Call “FrTp_ChangeParameter” and differs than from the configured default value.
No included containers	

7.4 Initialization and shutdown

FRISOTP1028: The FrIsoTp module shall have two internal states, `FRISOTP_OFF` and `FRISOTP_ON`.

FRISOTP1029: The FrIsoTp module shall implement a static status variable *`FrIsoTpState`* to denote whether the FrIsoTp module is initialized or not[9].

FRISOTP1030: The FrIsoTp module shall be in the `FRISOTP_OFF` state after power up.

FRISOTP1032: The FrIsoTp module shall change to the internal state `FRISOTP_ON` when the FrIsoTp has been successfully initialized by the service primitive `FrTp_Init()`.

FRISOTP1033: The FrIsoTp module shall performed normal FrIsoTp operation tasks (e.g. segmentation, reassembly etc.) only when the FrIsoTp module is in the `FRISOTP_ON` state[10].

[9] This variable is used for development error detection.

[10] This requires that FrTp_Init() is called before the normal FrIsoTp functionality is used by the COM-Stack.

FRISOTP1034: The service primitive `FrTp_Init` shall initialize all global variables of the module and sets all transport protocol connections in a sub-state of `FRISOTP_ON`, in which neither transmissions nor receptions are in progress.

FRISOTP1035: If the FrIsoTp module is in the global state `FRISOTP_ON`, a call of the service primitive `FrTp_Init` shall return the module to an uncritical idle state (idle state = `FRISOTP_ON`, but neither transmission nor reception are in progress) and the module shall loose all current connections.

FRISOTP1036: The FrIsoTp module shall change to the internal state `FRISOTP_OFF` when the service primitive `FrTp_Shutdown()` has been executed successfully.

FRISOTP1037: The FrIsoTp module shall raise an development error `FRISOTP_E_UNINIT` when

a) development error detection for the FrIsoTp module is enabled and

b) any function (except `FrTp_GetVersionInfo`) is called before the function `FrTp_Init` has been called.

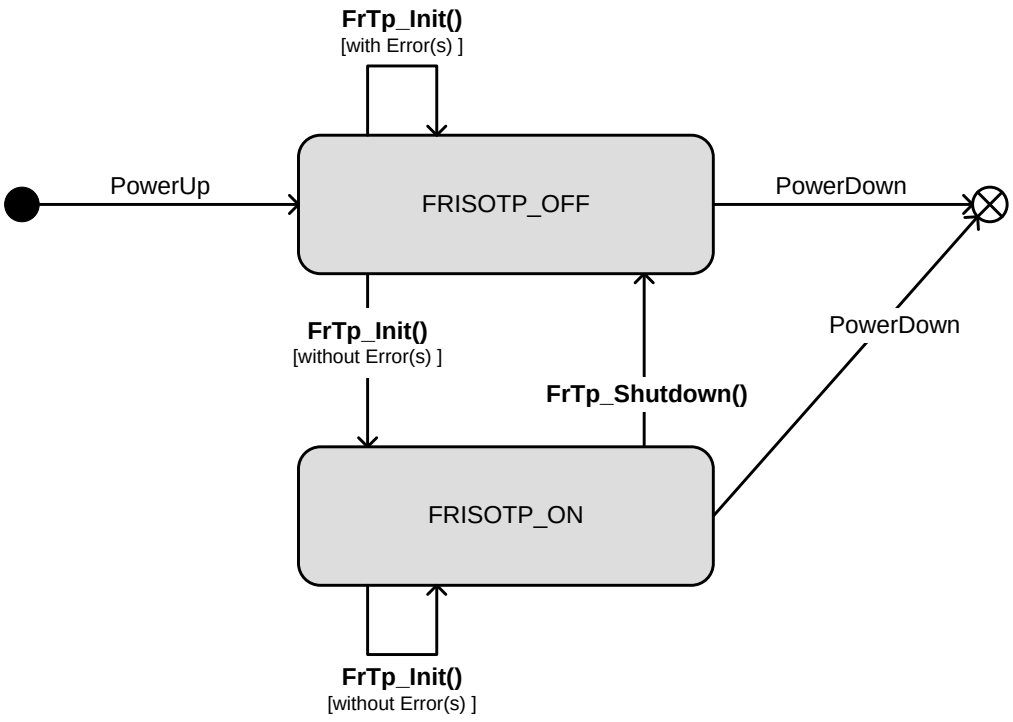


Diagram 2: FrIsoTp Initialization and shutdown state diagram

7.5 Data Transfer Processing

This chapter covers all topics of FrIsoTp module data transfer processing if transmission of data (Fr N-SDU) is requested by an upper layer (e.g. COM, DCM etc.) via PduR module or if data (Fr N-PDUs) have been received via FrIf module. For a better understanding the different topics are encapsulated in several sub-clauses starting with the basic definition of data transfer and reception. Buffer handling is described within an additional chapter.
The FlexRay protocol stack supports two different buffer access modes for data transmission:

a) Immediate Buffer Access Mode
b) Decoupled Buffer Access Mode

Due to this fact there are two different sequences for data transfer processing.

7.5.1 Flags

The FrIsoTp module uses several flags to signal internal states. (see also sequence diagrams in Chapter 9). This chapter describes the flags required for inter-module state handling.

7.5.1.1 TX_SDU_AVAILABLE

The `TX_SDU_AVAILABLE` flag is set by the service primitive *FrTp_Transmit* to indicate the N-SDU transmit request.

FRISOTP415: The `TX_SDU_AVAILABLE` flag shall exist for every channel.

FRISOTP416: The `TX_SDU_AVAILABLE` flag shall indicate an Fr N-SDU transmit request for a configured connection on an allocated channel.

Note: For detailed information about set and reset conditions and behaviour please refer to chapter 7.5.2 and **FRISOTP1057**.

7.5.1.2 TX_SDU_UNKNOWN_MSG_LENGTH

The `TX_SDU_UNKNOWN_MSG_LENGTH` flag is set by the service primitive *FrTp_Transmit* to indicate the N-SDU transmit request with an unknown message length. Depending on the status of that flag the FrIsoTp module will recall the service primitive *`PduR_FrTpProvideTxBuffer`* several times until all data are transmitted.

FRISOTP1101: The `TX_SDU_UNKNOWN_MSG_LENGTH` flag shall indicate an Fr N-SDU transmit request with unknown message length.

FRISOTP1102: The `TX_SDU_UNKNOWN_MSG_LENGTH` flag shall exist for every channel.

Note: For detailed information about set and reset conditions and behaviour please refer to chapter 7.5.2 and **FRISOTP1124**.

7.5.1.3 RX_PDU_AVAILABLE

The `RX_PDU_AVAILABLE` flag is set by the service primitive *FrTp_RxIndication* to indicate the reception of an N-PDU.

FRISOTP418: The RX_PDU_AVAILABLE flag shall exist for every Fr N-PDU, which is configured to be received by the FrIsoTp module.

Note: For detailed information about set and reset conditions and behaviour please refer to chapter 7.5.3 and **FRISOTP1074**.

7.5.1.4 RX_ERROR

The RX_ERROR[11] flag is required in case transmission with acknowledgement and retry is configured. During a segmented data reception an error could occur but sending FlowControl is currently not possible. In that case the information about an error has to be stored until sending a FlowControl is allowed.

FRISOTP428: The RX_ERROR flag shall exist for every `FrIsoTpChannel`.

FRISOTP429: The RX_ERROR flag shall indicate that an error occurred during a segmented reception.

FRISOTP430: The RX_ERROR flag shall be cleared after the reaction (Retry, Negative Acknowledgement, abortion).

7.5.2 Transmit Data

7.5.2.1 Transmit Data via ‘Immediate Buffer Access’ Mode

This chapter defines a data transfer requested by an upper layer (e.g. COM, DCM etc.) via ‘Immediate Buffer Access’ Mode.

[11] See ISO 10681-2 – chapter 6.5.7.2.3.

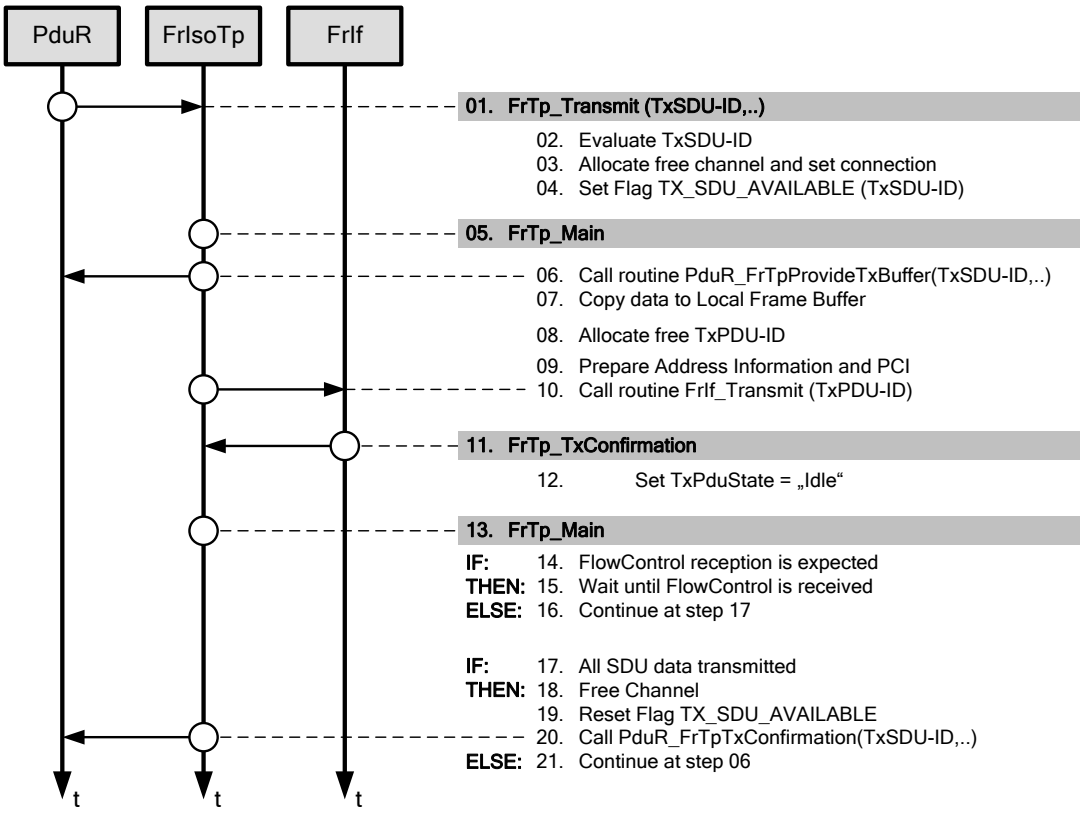


Figure 14: Transmit data overview in 'Immediate Buffer Access' mode

Figure 14 depicts the internal processing for data transmission in principle[12] if "Immediate Buffer Access" mode is configured[13]. Below there is a description of the different steps which are necessary to transmit data via FrIsoTp.

Step 1 - 4

FRISOTP136: Sending Fr N-SDU data shall always be initiated by the service primitive call of *FrTp_Transmit* (see chapter 8.3.3.1).

FRISOTP1043: The FrIsoTp module shall evaluate the value of PduInfoType.SduLength:

SduLength = 0: Transmission with unknown message length is requested
SduLength ≠ 0: Transmission with known message length is requested.

FRISOTP1044: If support unknown message length is configured the FrIsoTp module shall set the flag `TX_SDU_UNKNOWN_MSG_LENGTH` according to the result of **FRISOTP1043** (see also chapter 7.5.1.2).

FRISOTP1134: The FRISOTP module shall raise an development error FRISOTP_E_UMSG_LENGTH_ERROR when
a) a transmission with unknown message length is detected and

[12] Figure 14 depicts only the overview of data transmission for a single channel without further functionality (e.g. transmit cancellation) or internal behaviour (e.g. internal control data handling, return values etc.). Also a schedule for data transfer in concurrent channels is not described here.
[13] Buffer access mode is configured for each N-PDU and is referenced via the PDU-Pool.

b) support of unknown message legth is not configured and
c) development error detection is enabled for the FrIsoTp module.

The service primitive parameter `FrIsoTpTxSduId` shall be used to select the correct connection. This shall be done by searching for the correct entry within the `FrIsoTpConnection` container (`FrIsoTpConnection.FrIsoTpTxSduId`). If a valid parameter `FrIsoTpTxSduId` is given, the FrIsoTp module shall search for a free channel resource (`FrIsoTpTxChannelState = Idle`) and allocate them to control the requested Tx data transfer.

FRISOTP1038: An ongoing data transfer shall be signalled by the flag `TX_SDU_AVAILABLE` (see also chapter 7.5.1.1).

FRISOTP1039: The service primitive shall set the flag `TX_SDU_AVAILABLE` only, if
a) the requested `FrIsoTpTxSduId` is valid
b) a free channel resource is available (`FrIsoTpTxChannelState = Idle`)

FRISOTP1040: If the current parameter `FrIsoTpTxSduId` is not supported the service primitive `FrTp_Transmit`
a) shall be terminated and the return value shall be set to E_NOT_OK (see also chapter 8.2.1) and
b) the FrIsoTp module shall raise an development error FRISOTP_E_INVALID_PDU_SDU_ID when development error detection for the FrIsoTp module is enabled.

FRISOTP1041: If no free channel is available (`FrIsoTpTxChannelState ≠ Idle`) the service primitive `FrTp_Transmit` shall be terminated and the return value shall be set to E_NOT_OK (see also chapter 8.2.1)[14].

FRISOTP1185: The FrIsoTp module shall raise an development error FRISOTP_E_NO_CHANNEL when development error detection for the FrIsoTp module is enabled.

Step 5 - 10

If the `TX_SDU_AVAILABLE` flag is set, the FrIsoTp module has to request Tx data which has to be transmitted by calling the service primitive `PduR_FrTpProvideTxBuffer()`. After receiving Tx data and length information, copy the data to local frame buffer. With knowledge of the available data size FrIsoTp module scans the `FrIsoTpTxPduPool` and allocates the first free `FrIsoTpPdu` for that data transfer. Depending on the available FrIsoTp-N-SDU length and the `FrIsoTpTxPduPool's` free `FrIsoTpPdu` length the FrIsoTp module decides whether segmentation is necessary or not for that N-SDU transfer.. In a next step the

[14] This scenario could occur on gateways, if communication via more connections is requested than channels resources are configured.

corresponding Address Information and PCI are prepared and the service primitive `FrIf_Transmit` is called with the corresponding `FrIsoTp_TxPduId`.

FRISOTP1042: If the `TX_SDU_AVAILABLE` flag is set, the FrIsoTp module shall call the service primitive `PduR_FrTpProvideTxBuffer()` to get the currently available FrIsoTp N-SDU Tx data and Length information.

FRISOTP1045: The FrIsoTp module shall always allocate the first free `FrIsoTpTxPdu` while scanning the corresponding `FrIsoTpTxPduPool` (see also chapter 7.3.2.2).

FRISOTP1046: If a free `FrIsoTpTxPdu` is identified, the FrIsoTp module shall use this `FrIsoTpTxPdu` to continue current transmission process.

FRISOTP1047: If no free `FrIsoTpTxPdu` is identified, the FrIsoTp module shall stop processing for the corresponding connection within the current task.

FRISOTP1048: The FrIsoTp module shall decide whether segmentation for the requested N-SDU transfer is required or not depending on the length information of the first allocated `FrIsoTpTxPdu` from an `FrIsoTpTxPduPool` for the currently processed `FrIsoTpConnection` (see also Diagram 3).

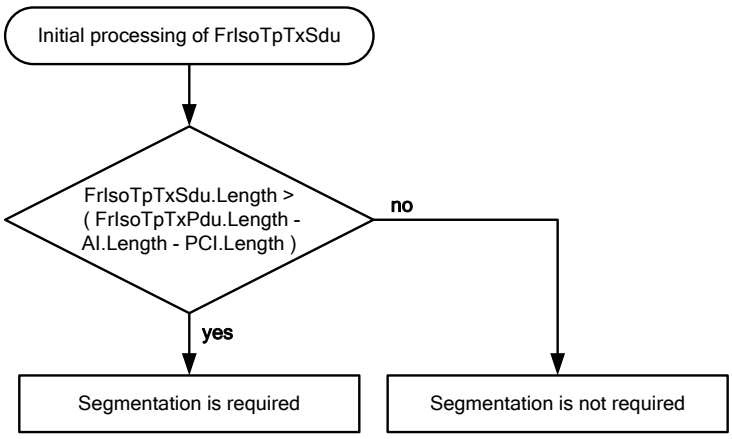


Diagram 3: Segmentation decision

Note: The decision whether segmentation is possible or not depends also on the connection mode (1:1 or 1:n). Please refer to chapter 7.5.2.4.

FRISOTP1123: The FrIsoTp module shall call the service primitive `PduR_FrTpProvideTxBuffer()` to get the currently available FrIsoTp N-SDU data with a length of `FrIsoTpTxPdu` length to the corresponding transmit data.

FRISOTP1188: If the service PduR_FrTpProvideTxBuffer does not provide all data bytes of the N-SDU during the first call, it shall be called again and again until the whole N-SDU has been transmitted.

FRISOTP1049: The FrIsoTp module shall prepare the Address Information and PCI according to the result of **FRISOTP1048** as defined in specification ISO 10681-2 [16].

FRISOTP1050: The FrIsoTp module shall initiate an N-PDU data transfer by calling the service primitive `FrIf_Transmit`() with the `FrIsoTpTxPduId` FrIsoTpTxPduId of the recently allocated FrIsoTpTxPdu.

FRISOTP1051: The FrIsoTp shall set the corresponding data length referenced by the service primitive `FrIf_Transmit`'s parameter `PduInfoType` to the exact data length of the buffer[15].

Step 11 - 12

If the N-PDU was successfully transmitted by the FrIf module, the FrIf module shall call the service primitive `FrTp_TxConfirmation`. Within this service primitive the FrIsoTp module shall reset the state of the corresponding `FrIsoTpTxPdu`.

FRISOTP1052: The service primitive `FrTp_TxConfirmation` shall be called by the underlying layer module after a successful transmission of the corresponding N-PDU.

FRISOTP1053: The service primitive `FrTp_TxConfirmation` shall be called by the underlying layer module with the corresponding `FrIsoTpTxPduId` of the successful transmitted PDU.

FRISOTP1054: The service primitive `FrTp_TxConfirmation` shall reset the state of the corresponding `FrIsoTpTxPdu` (N-PDU) to "`idle`".

Step 13 - 16

Depending on ISO-10681-2 protocol handling in some cases a response N-PDU (Flow Control) from the receiver is expected by the sender. Hence the sender has to wait until the response N-PDU (Flow Control) is received and continue processing after reception.

FRISOTP1055: The FrIsoTp module shall implement a timing and timeout behaviour as defined in chapter 7.5.8

Step 17 - 21

[15] FrTp transmits always the real amount of data stored in the corresponding buffer. FrTp is not responsible for fill bytes. Fill up N-SDUs to a configured frame size is done within lower layers (e.g. FrIf or FlexRay Driver). The FrTp only decides whether segmentation is necessary or not and to segment N-PDU Consecutive Frames to the maximum length of the corresponding PDU of the PduPool.

If all N-SDU data are transmitted the FrIsoTp module shall free all allocated resources and reset all flags which signals an ongoing data transfer for this connection.
If the data transmission is pending, the FrIsoTp module shall continue the data transfer at step 6.

FRISOTP1056: The FrIsoTp module shall free the allocated channel (`FrIsoTpTxChannelState = Idle`) if
a) all Tx N-SDU data are transmitted and
b) FrTp_TxConfirmation was given and
c) the final acknowledge is received in case acknowledge is configured.

FRISOTP1057: The FrIsoTp module shall reset the flag `TX_SDU_AVAILABLE`, if:
a) all N-SDU data are transmitted and
b) FrTp_TxConfirmation was given and
c) the final acknowledge is received in case acknowledge is configured.

FRISOTP1124: The FrIsoTp module shall reset the flag `TX_SDU_UNKNOWN_MSG_LENGTH`, if:
a) all N-SDU data are transmitted and
b) FrTp_TxConfirmation was given and
c) the final acknowledge is received in case acknowledge is configured or
d) if transmission was aborted and FrTp_TxConfirmation was given.

FRISOTP1058: The FrIsoTp module shall call the service primitive `PduR_FrTpTxConfirmation` for the corresponding `FrIsoTpTxSduId` if
a) all N-SDU data are transmitted and
b) FrTp_TxConfirmation was given
c) the final acknowledge is received in case acknowledge is configured.

7.5.2.2 Transmit Data via 'Decoupled Buffer Access' Mode

Figure 15 depicts the internal processing for data transmission in principle[16] if "Decoupled Buffer Access" mode is configured[17]. Below there is a description of the different steps which are necessary to transmit data via FrIsoTp.

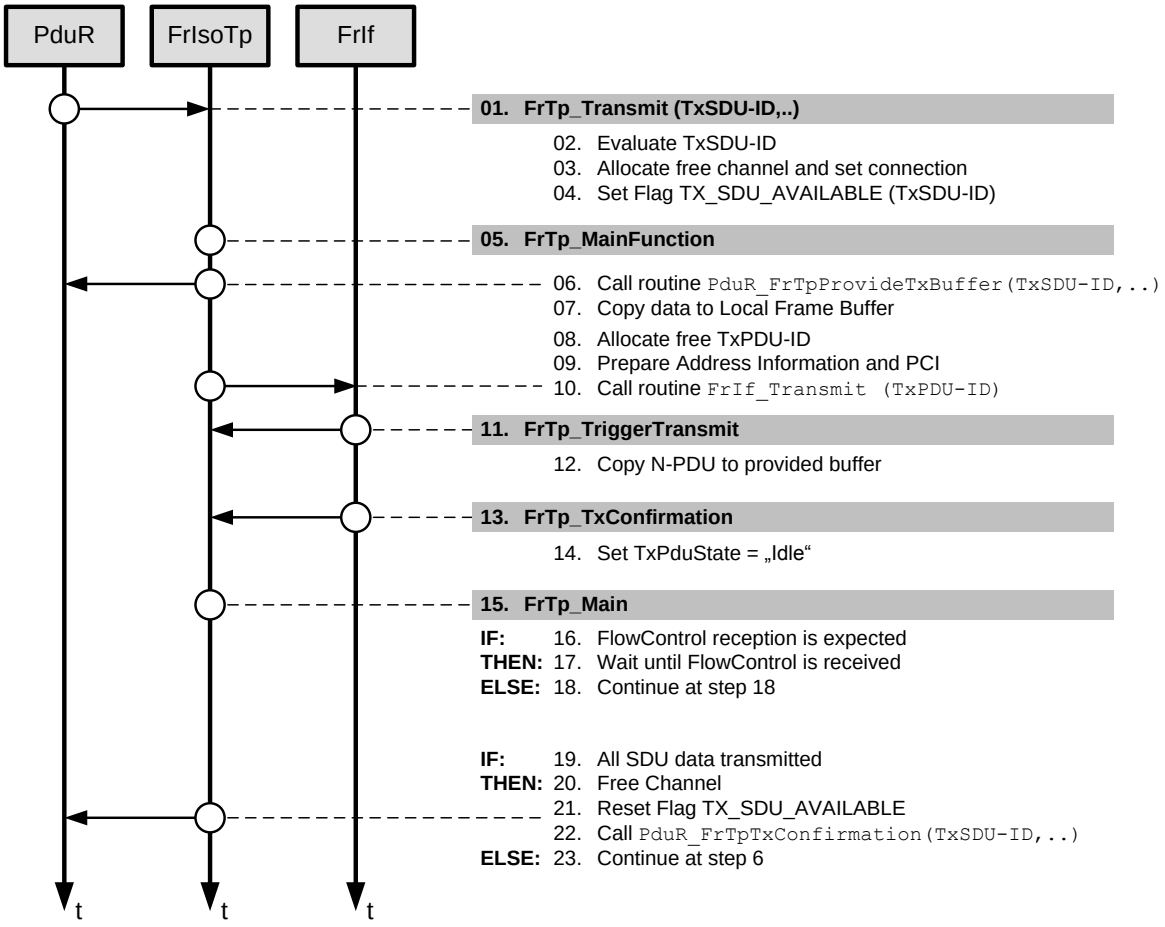


Figure 15: Transmit data overview in 'Decoupled Buffer Access' mode

Step 01 - 10

Step 01 - 10 in 'decoupled access mode' are equal to step 01 – 10 in 'immediate access mode'. Please refer chapter 7.5.2.1.

For step 09 it is recommended to set the address information and PCI to a local buffer because the service primitive *FrTp_TriggerTransmit* is called in interrupt mode and therefore the processing time to copy the complete N-PDU (Address Information, PCI and (part of) SDU data) shall be as short as possible.

[16] Figure 15 depicts only the overview of data transmission for a single channel without further functionality (e.g. transmit cancellation) or internal behaviour (e.g. internal control data handling, return values etc.). Also a schedule for data transfer in concurrent channels is not described here.

[17] Buffer access mode is configured for each N-PDU (refer to FrIf) and is referenced via the PDU-Pool.

Step 11 - 12

FRISOTP1059: The service primitive *FrTp_TriggerTransmit* shall be called by the FrIf module to propagate FrIsoTp N-PDUs to the lower layers (e.g. FlexRay Driver).

FRISOTP1060: The service primitive *FrTp_TriggerTransmit* shall copy the Address Information, PCI and N-SDU data to the corresponding buffer, which is referenced by the service primitive parameter `PduInfoType`.

FRISOTP1061: The service primitive *FrTp_TriggerTransmit* shall set the corresponding data length referenced by the service primitive parameter `PduInfoType` to the exact data length of the buffer.

Step 13 - 23

Steps 13 - 23 in 'decoupled access mode' are equal to step 10 – 20 in 'immediate access mode'. Please refer chapter 7.5.2.1.

7.5.2.3 Data Transfer with unknown message length

ISO10681-2 supports the possibility to transmit data with an unknown message length.

FRISOTP1062: The functionality to support data transmission with unknown message length shall be configurable by compiler switch.

FRISOTP1063: If a 1:n connection is configured (parameter `FrIsoTpMultipleReceiverCon` is set), a Data transfer with unknown message length shall not be processed[18] and the service primitive call *FrTp_Transmit* shall be rejected with the return value E_NOT_OK.

FRISOTP1187: The FrIsoTp module shall raise an development error FRISOTP_E_SEG_ERROR when
a) development error detection for the FrIsoTp module is enabled and
b) a 1:n connection is requested as described in **FRISOTP1061**.

[18] Unknown message length data transfer requires segmentation because at least a StartFrame and a LastFrame have to be transmitted. Segmentation of 1:n connections is not allowed (see also chapter 7.5.2.4)

FRISOTP1064: An upper layer's data transmission with unknown message length shall be initiated by an calling the service primitive FrTp_Transmit with the service primitive parameter `PduLength` = 0 ('zero')

FRISOTP1065: During an ongoing data transfer with unknown message length the service primitive parameter `Length` of the service primitive `PduR_FrTpProvideTxBuffer()` shall be set to the zero that means the transmit data can be of arbitrary size .

FRISOTP1067: The FrIsoTp module shall add all `PduInfoType.PduLength` values to calculate the total message length which is transmitted by the LastFrame (LF).

7.5.2.4 Segmentation condition for data transfer

FrIsoTp module provides 1:1 connections as well as 1:n connections. According to ISO10681-2, the FrIsoTp module shall refuse segmented 1:n connections. Due to the possibility of different PDU lengths within an FrIsoTpTxPduPool the scenario of segmented 1:n connections could occur and shall be solved by the requirement(s) specified within this chapter.

Note: There shall be a configuration check to ensure that only single frames (unsegmented N-SDUs) can use the fan-out i.e. any 1:n connection shall exclusively reference single frame N-SDUs.

7.5.3 Receive Data

This chapter defines a data reception on FrIsoTp module requested by the lower layer FlexRay Interface (FrIf).

Figure 16 depicts the internal processing for data reception in principle[19]. Below there is a description of the different steps which are necessary to receive data via FrIsoTp.

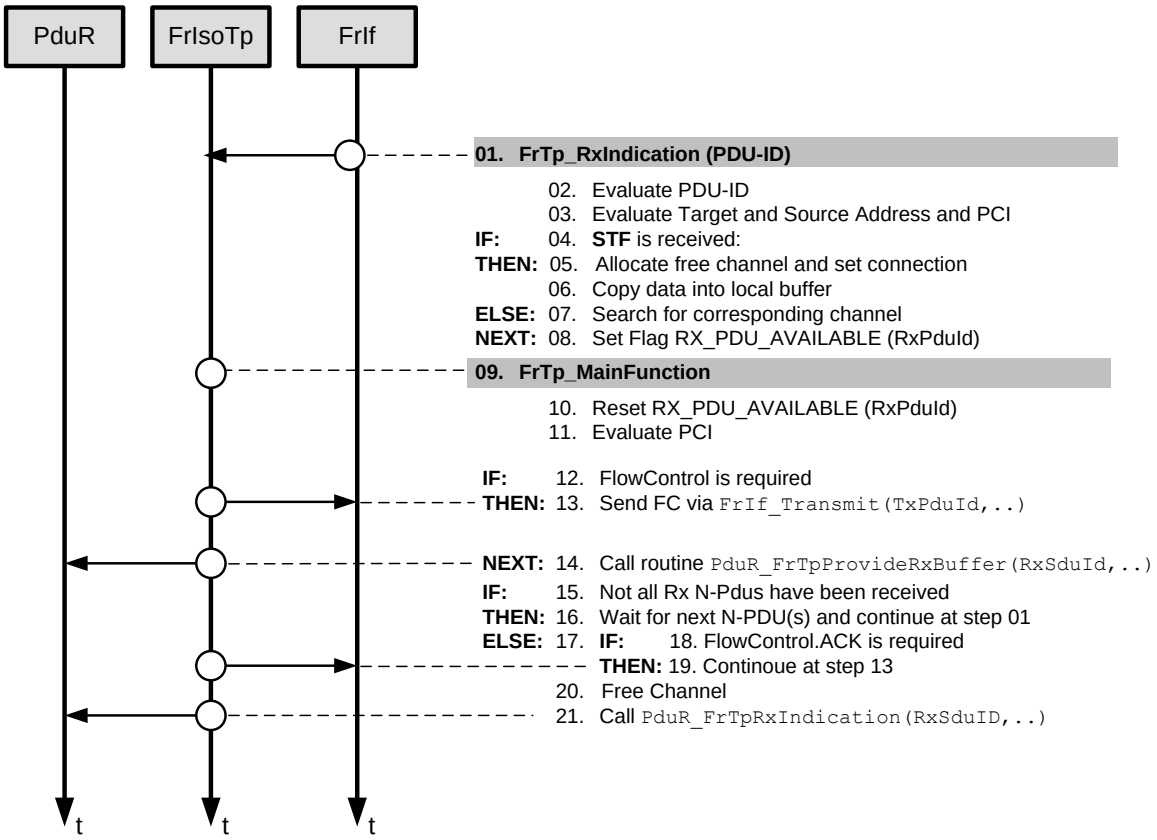


Figure 16: Receive data overview

Step 1 - 8

FRISOTP137: Receiving shall be initiated by the service primitive call *FrTp_RxIndication.*

The FrIsoTp module shall validate the `FrIsoTpRxPduId`. If an invalid `FrIsoTpRxPduId` is received, the service primitive FrTp_RxIndication is terminated. For a valid `FrIsoTpRxPduId` the FrIsoTp module evaluates the target and source address and selects the corresponding `FrIsoTpConnection`. If a Startframe (STF)

[19] Figure 16 depicts only an overview of data reception for a single channel without further functionality (e.g. transmit cancellation) or internal behaviour (e.g. internal control data handling, return values etc.). Also a schedule for data transfer in concurrent channels is also not described here.

is received a free FrIsoTpChannel resource (FrIsoTpRxChannelState = Idle) has to be allocated for that connection.
If a consecutive frame (CF) or a last frame (LF) has been received, the corresponding channel has to be evaluated and the Rx_PDU_AVAILABLE flag shall be set.

FRISOTP1069: The FrIsoTp module shall process a received FrIsoTp N-PDU only if
a) a valid FrIsoTpRxPduId is received and
b) the FrIsoTp N-PDU's address information matches to the configured FrIsoTpConnection address information.

FRISOTP1070: If an invalid (undefined) FrIsoTpRxPduId is received the FrIsoTp module shall
a) ignore the FrIsoTp N-PDU and
b) shall raise an development error FRISOTP_E_INVALID_PDU_SDU_ID when development error detection for the FrIsoTp module is enabled.

FRISOTP1071: A matching FrIsoTpConnection is only identified if
c) the received FrIsoTp N-PDU's "Target Address" (see ISO 10681-2) is equal to the configured FrIsoTpConnection's Local Address (FrIsoTpLa, see section 10.2) and
d) the received FrIsoTp N-PDU's "Source Address" (see ISO 10681-2) is equal to the configured FrIsoTpConnection's Remote Address (FrIsoTpRa, see section 10.2).

FRISOTP1072: If the address check doesn't match to any configured FrIsoTpConnection the received FrIsoTp N-PDU shall be ignored.

FRISOTP1074: The service primitive shall set the flag RX_PDU_AVAILABLE only, if
a) the requested FrIsoTpRxPduId is valid and
b) the address check matches to a configured FrIsoTpConnection and
c) a free channel resource is available (FrIsoTpRxChannelState = Idle)

FRISOTP1075: If the current parameter FrIsoTpRxPduId is not supported the service primitive FrTp_RxIndication shall be terminated without any further action.[20]

FRISOTP1076: If no free channel is available (FrIsoTpRxChannelState ≠ Idle) the service primitive FrTp_RxIndication shall be terminated without any further action.

[20] If DET is active a corresponding error shall be set.

FRISOTP1186: The FrIsoTp module shall raise an development error FRISOTP_E_NO_CHANNEL when development error detection for the FrIsoTp module is enabled

FRISOTP1077: Within the service primitive *FrTp_RxIndication* the FrIsoTp module shall copy the received StartFrame PDU into a local buffer[21].

Step 9 - 13

According to ISO 10681-2 protocol (evaluate PCI) it is possible that a received N-PDU requires an N-PDU response (e.g. FlowControl). In that case the FrIsoTp module shall allocate the first free N-PDU from the referenced PDU pool, prepare the response and initiate the transmission process by a service primitive call *FrIf_Transmit* with the corresponding FrIsoTpTxPduId. After transmission the FrIsoTp module shall wait for reception of consecutive N-PDUs. If no N-PDU response is required by protocol the FrIsoTp module shall continue reception handling.

FRISOTP1080: If transmission of an N-PDU response is required by ISO10681-2 protocol handling, the FrIsoTp module shall send the corresponding N-PDU (e.g. FlowControl) to the initial sender node.

Step 14

FRISOTP1079: The FrIsoTp module shall extract the N-SDU data from the received N-PDU data according to ISO 10681-2

FRISOTP1138: The FrIsoTp module shall get the Rx buffer to transer the received N-SDU (fragment) by calling the service primitive PduR_FrTpProvideRxBuffer [22] and copy the received N-SDU (fragment) to the received RX buffer.

FRISOTP421: The RX_PDU_AVAILABLE flag shall be cleared when finished processing the Fr N-PDU.

Step 15 - 16

The FrIsoTp module could calculate whether all N-PDUs of an N-SDU are received. If the communication is still ongoing the FrIsoTp module shall continue data reception at step 01.

[21] Only the received StartFrame PDU shall be stored temporarry in a local buffer. This is necessray in case of a gateway has temporarry no free resources to process that frame. The correct protocol and timing behavour is ensured if FrIsoTp sends a FlowControl PDU after a free channel was allocated.
[22] The procedure is also used for the “Routing-On-The-Fly” behaviour for gateways.

Step 17 - 21

If all FrIsoTp Rx-PDUs of a complete N-SDU transmission have been received the FrIsoTp module shall send an Acknowledgement if required and free the allocated channel ressource. In a next step the FrIsoTp module shall call the service primitive `PduR_FrTpRxIndication` with the corresponding `FrIsoTpRxSduId` to signal upper layers that an N-SDU has been received.

FRISOTP1081: The FrIsoTp module shall free the allocated channel (`FrIsoTpRxChannelState = Idle`) if
a) all N-SDU data are received and
b) all required response N-PDUs (FlowControl) are transmitted and TxConfirmation was given.

FRISOTP1083: The FrIsoTp module shall call the service primitive *`PduR_FrTpRxIndication`* if
a) all N-SDU data are received and
b) all required response N-PDUs (FlowControl) are transmitted and TxConfirmation was given.

7.5.3.1 Receive Cancellation

FRISOTP1180: If development error detection is enabled:
The function FrTp_CancelReceive shall check the parameter FrIsoTpRxSduId for being valid. If the check for FrIsoTpRxSduId fails, the function shall raise the development error `FRISOTP_E_INVALID_PDU_SDU_ID` and return E_NOT_OK.

FRISOTP1181: The FrIsoTp shall abort the reception of the current N-SDU if the service FrTp_CancelReceive provides a valid FrIsoTpRxSduId.

FRISOTP1182: The FrIsoTp shall reject the request for receive cancellation in case of an
a) unsegmented reception or
b) in case the FrIsoTp is in the process of receiving the LastFrame of the N-SDU and shall return E_NOT_OK.

FRISOTP1183: If the FrTp_CancelReceive service has been successfully Executed, FrTp_CancelReceive shall return with result E_OK.

7.5.3.2 Receive with unknown message length

Length (greater than zero) of received message should be passed mandatorily to service primitive PduR_FrTpProvideRxBuffer. So FrIsoTp module does not support receive data with unknown message length according to ISO 10681-2.

7.5.4 Buffer Handling

7.5.4.1 Buffer Access Mode

FRISOTP1084: For Tx direction the FlexRay ISO Transport Protocol Layer shall support
a) "Immediate Buffer Access" mode and
b) "Decoupled Buffer Access" mode.

7.5.4.2 Request for Buffer

The FrIsoTp module does not provide message buffers, neither for sending nor for receiving. Instead the FrIsoTp module works directly on the memory area of the upper layers (e.g. PduR, DCM, or COM). To access these memory areas, the FrIsoTp module uses the indicator returned by the service primitive PduR_FrTpProvideTxBuffer *and* PduR_FrTpProvideRxBuffer.

7.5.4.2.1 Request for RxBuffer

FRISOTP1189: First call to the function PduR_FrTpProvideRxBuffer on reception of a (un)segmented Start Frame (STF) shall be succeeded. In case of failure; retry shall not be performed and therefore the connection is aborted then the PduR_FrIsoTpRxIndication shall be notified with TFRSLT_E_ABORT. For an acknowledgement frame FlowControl.ABT shall be sent back to the sender.

If the call for an RxBuffer (service primitive PduR_FrTpProvideRxBuffer) does not provide a valid buffer, the FrIsoTp module shall react as listed below:

FRISOTP405: If service primitive's PduR_FrTpProvideRxBuffer return BufReq_ReturnType value as BUFREQ_E_NOT_OK, then FrIsoTp module shall
1) abort reception and
2) send an N-PDU FlowControl.ABT.

FRISOTP1160: If service primitive's PduR_FrTpProvideRxBuffer returns a valid buffer but this buffer is not large enough for the next block, then the FrIsoTp module shall
1) invoke PduR_FrTpProvideRxBuffer repeatedly again
2) shall send an N-PDU FlowControl.WAIT each time before FRISOTP_TIME_BR times out until FrIsoTpMaxFcWait (see section 10.2) is reached

Note: Repeated calls to PduR_FrTpProvideRxBuffer shall be performed in FrIsoTp_MainFunction.

FRISOTP1161: If service primitive's PduR_FrTpProvideRxBuffer return BufReq_ReturnType value as BUFREQ_E_OVFL, then FrIsoTp module shall
1) abort reception and
2) shall send an N-PDU FlowControl.OVFL.

FRISOTP412: The FrIsoTp module shall use an N-PDU FlowControl parameter <Buffer Size> depending on the size of the provided RxBuffer (returned by the service primitive call *or PduR_FrTpProvideRxBuffer)*, to ensure that the transferred data bytes within an upcoming block could be stored into the provided receive buffer.

FRISOTP1190: FrIsoTp module shall call the function PduR_FrTpProvideRxBuffer after copying the segmented Start Frame (STF) to get the next block of data for the following CFs and in each call (except first call) the function PduR_FrTpProvideRxBuffer shall return size of the data actually stored in the buffer.

7.5.4.2.2 Request for TxBuffer

FRISOTP402: If the call for a Tx data (service primitive *PduR_FrTpProvideTxBuffer*) does not provide a valid Tx data length and if service primitive notification result type value is *BUFREQ_E_BUSY*, then FrIsoTp module shall try up to *FrIsoTpMaxFcWait* times to get a valid Tx data length.

FRISOTP1162: If the call for a Tx data (service primitive *PduR_FrTpProvideTxBuffer*) does not provide a valid Tx data length and if service primitive notification result type value is *BUFREQ_E_NOT_OK*, then FrIsoTp module shall abort the transfer.

7.5.5 Dynamic Bandwidth Assignment

From FrIsoTp's point of view physical FlexRay bandwidth is represented by N-PDUs. As depict in Figure 17 there is a direct mapping between N-PDUs and L-PDUs (done within FrIf module's frame construction plan).

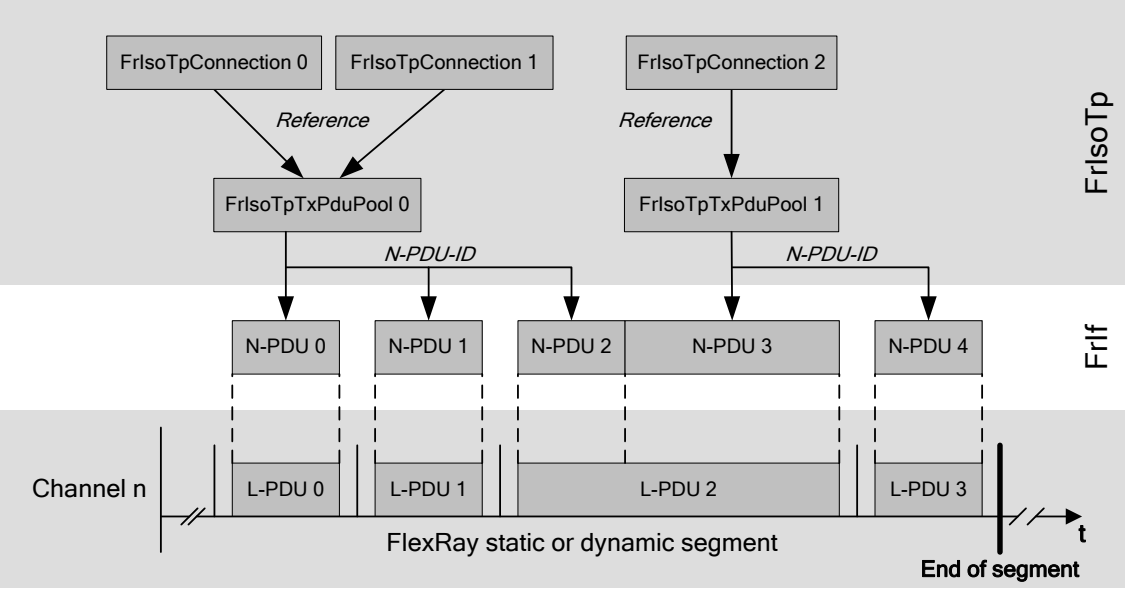


Figure 17: Mapping of N-PDUs to N-PDU-Pools

An FrIsoTpTxPduPool could be referenced by different `FrIsoTpConnections` (see also chapter 7.3.2.2). Depending on the number of currently active `FrIsoTpConnections` the bandwidth (N-PDUs) is shared between them[23]. By supporting dynamic bandwidth assignment, a support of different communication scenarios is possible.
Figure 18 depicts two different scenarios which could be supported with only one FrIsoTp configuration. From gateway's point of view different communication scenarios are possible:

a) single connection communication
   Complete bandwidth (slots) is assigned to one communication link (e.g. to ECU 2)
b) multiple connection communication
   Bandwidth (slots) is shared between different communication links to different ECUs (e.g. ECU 1-3).

[23] Scenario: e.g. gateway communication: For diagnostic communication it is necessary to define a connection to each ECU. In some cases it is required to have a maximum communication in parallel on the other hand it is required to have maximum bandwidth to exactly on ECU (e.g. reprogramming purpose). If dynamic bandwidth assignment is possible, both scenarios are educible with a minimum amount of FlexRay resources ("slots").

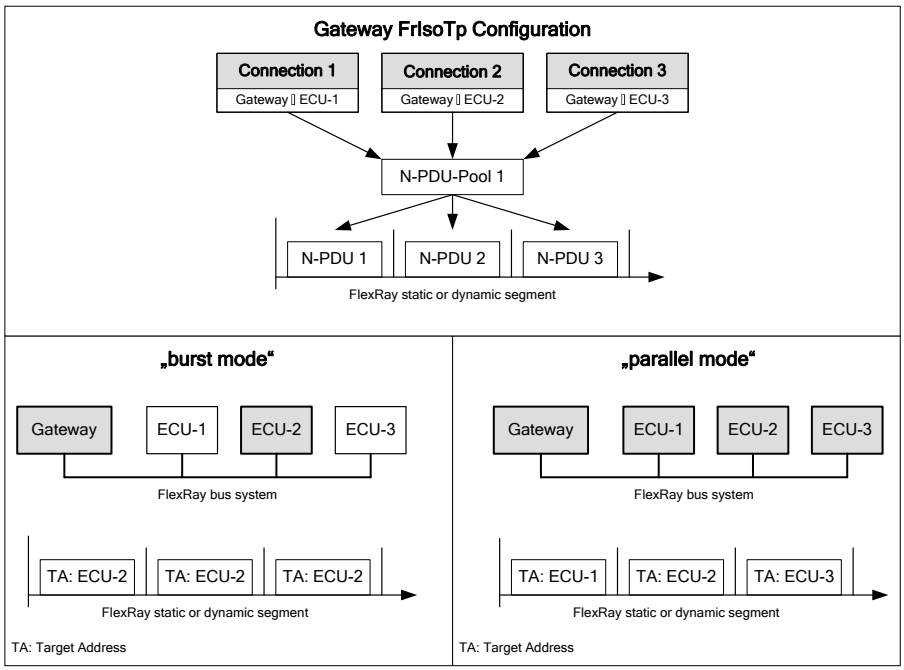


Figure 18: PDU-Pool sharing by different connections

The connection handler controls the partitioning of bandwidth (see Figure 19).

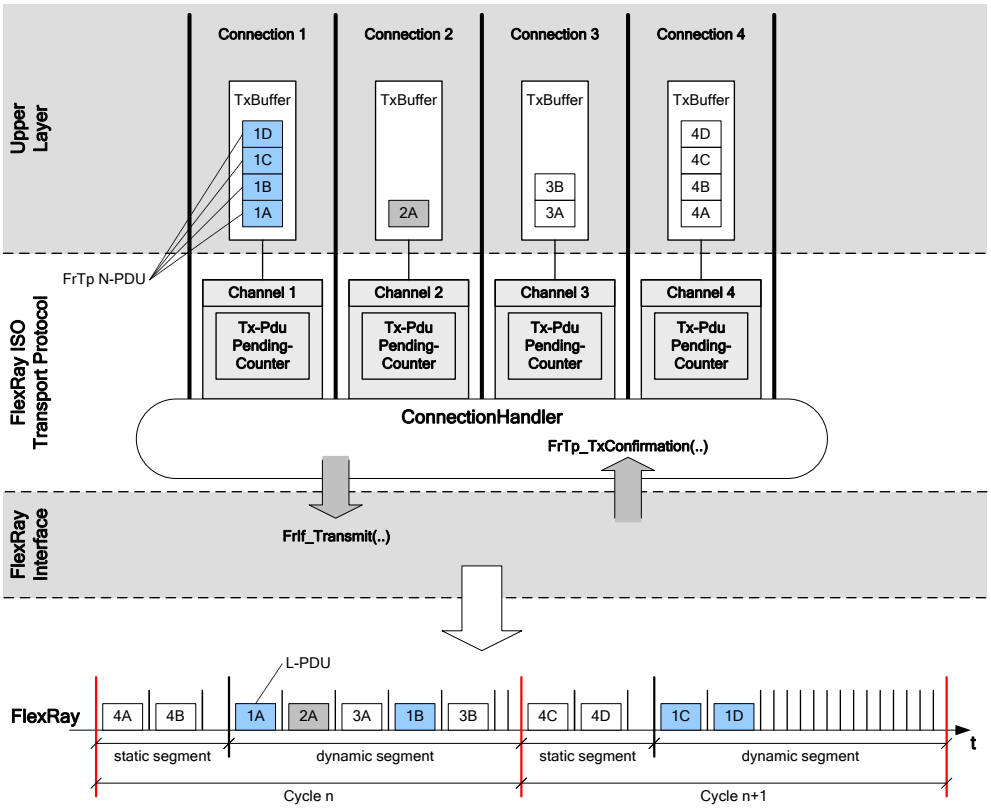


Figure 19: Connection Handler for different connections

The bandwidth assignment can change each communication cycle depending on the active communication links. Especially a gateway could have multiple active communication links in parallel. Hence there are some additional requirements for the FrIsoTp module to handle concurrent connections. Especially for a segmented data transfer it is necessary to ensure that the connection handler could not swap the order of consecutive frames.[24]

FRISOTP1088: All FrIsoTp Tx N-PDUs within an `FrIsoTpTxPduPool` shall be listed in ascending order depending on their position within the global N-PDU network plan[25].

Note: See also chapter 7.3.2.2

FRISOTP1089: Each FrIsoTp Tx N-PDU within an `FrIsoTpTxPduPool` could have an individual length.[26]

[24] This could occure within the dynamic segment if the transfer of the last L-PDU (including a consecutive frame) is skipped for the current communication cycle and within the next communication cycle other consecutive frames are sent in front of the skiped one.

[25] ECU specific N-PDU plan means that each N-PDU (uniquely identified by its N-PDU-ID) is mapped to an L-PDU. Each L-PDU is uniquely identified by its parameter set “slot-ID”, “cycle counter” and “cycle offset”. Hence all N-PDUs have an implicit order too.

If more than one FrIsoTp Tx N-PDU is used for data transmission within one connection the number of currently used FrIsoTp Tx N-PDUs has to be controlled. Hence a counter is defined to track all initiated but currently not confirmed FrIsoTp Tx N-PDU transmissions.

FRISOTP1090: Each FrIsoTpChannel shall implement a runtime variable `TxPduPendingCounter` (see chapter 7.3.3.2).

FRISOTP1091: The `TxPduPendingCounter` shall be incremented each time the service primitive `FrIf_Transmit` was terminated with the return value `E_OK` for the corresponding FrIsoTp Tx N-PDU (`FrIsoTpTxPduId`).

FRISOTP1092: The `TxPduPendingCounter` shall be decremented each time the service primitive `FrTp_TxConfirmation` was called with the corresponding parameter `FrIsoTpTxPduId`.

FRISOTP1093: A TxConfirmation shall be given for each transmitted N-PDU by the underlying layer module by calling the corresponding service primitive `FrTp_TxConfirmation` with the corresponding `FrIsoTpTxPduId`.

The communication handler task shall process an active `FrIsoTpConnection` (referenced by an FrIsoTpChannel) only if the corresponding `TxPduPendingCounter` is zero at begin of the task. If the `TxPduPendingCounter` is unequal to zero an FrIsoTp Tx N-PDU confirmation is pending and the processing for the corresponding `FrIsoTpConnection` is skipped for the current communication handler task.

FRISOTP1094: An active `FrIsoTpConnection` (referenced by `FrIsoTpChannel`) shall only processed if the `TxPduPendingCouter` of the corresponding `FrIsoTpChannel` is zero (“0”) at begin of a communication handler task.

FRISOTP1095: If the `TxPduPendingCouter` is unequal to zero (“0”) the processing for the corresponding `FrIsoTpConnection` shall skipped for the current communication handler task.

FRISOTP1096: A communication handler task shall process all active `FrIsoTpConnections` alternately[27] as long as free FrIsoTp Tx N-PDUs are available within the referenced `FrIsoTpTxPduPool`.

---

[26] As depict in Figure 17, at the end of a segment it could occure that only an L-PDU with less payload could be placed in the schedule. Hence the mapped FrIsoTp N-PDU should have the corresponding length to prevent waste of bandwidth.

[27] Alternate means that a schedule has to be implemented which process all active FrIsoTpConnections. It is recommandet to use a simple round-robin method but other schedules are also possible.

7.5.6 Transmit Cancellation

According to ISO 10681-2 the FrIsoTp module supports “Transmit Cancellation” for an ongoing FrIsoTp N-SDU transfer. This functionality could disable by a global compiler switch.

FRISOTP1097: The “Transmit Cancellation” feature shall be (de)activated by static configuration of the FrIsoTp parameter *FrIsoTpTransmitCancellation* (see section 10.2).

FRISOTP384: A Transmit Cancellation shall be done by the call of the service primitive *FrTp_CancelTransmit()* (see FRISOTP150).

FRISOTP1116: When a transmission is still in progress, FrTp_CancelTransmit shall stop the transmission and shall return E_OK. When a connection is not active, or when the last N-PDU of a transmission without acknowledgement has already been forwarded to the FrIf, FrTp_CancelTransmit shall return E_NOT_OK.

FRISOTP385: If the transmit request is pending but the transmission has not started, *FrTp_CancelTransmit* (see FRISOTP150) shall immediately free the connection.

7.5.6.1 Transmit Cancellation for unsegmented data transfer

A Transmit Cancellation request for an unsegmented data transfer could occur on two different positions within FrIsoTp module's processing:

a) Before sending the StartFrame (STF)
   The Transmit Cancellation Request is effective.
b) After sending the StartFrame (STF)
   The Transmit Cancellation Request is not effective because of the StartFrame was sent.

Figure 20 depicts the transmit cancellation behavior of an unsegmented data transfer.

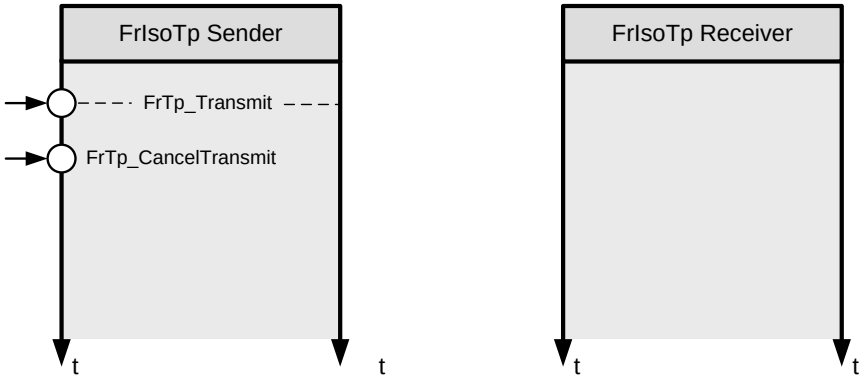


Figure 20: Transmit Cancellation at unsegmented data transfer

If a requested but currently not started data transfer shall be cancelled the service primitive *FrTp_CancelTransmit()* is called. *FrTp_CancelTransmit()* shall cancel the requested data transfer. On receiver side no data transfer is recognized.

7.5.6.2 Transmit Cancellation for segmented data transfer

A Transmit Cancellation request for a segmented data transfer could occur on three different positions within FrIsoTp module's processing:

a) Before sending the StartFrame (STF)
   The Transmit Cancellation Request is effective.
b) Within an ongoing data transfer
   The Transmit Cancellation Request is effective.
c) After sending the LastFrame (LF)
   The Transmit Cancellation Request is not effective because of because after having transmitted the LastFrame (LF) the transmission is finished

Figure 21 depicts the transmit cancellation behavior of a segmented data transfer. If a requested but currently not started data transfer shall be cancelled the service primitive *FrTp_CancelTransmit* is called. On receiver side no data transfer is recognized.

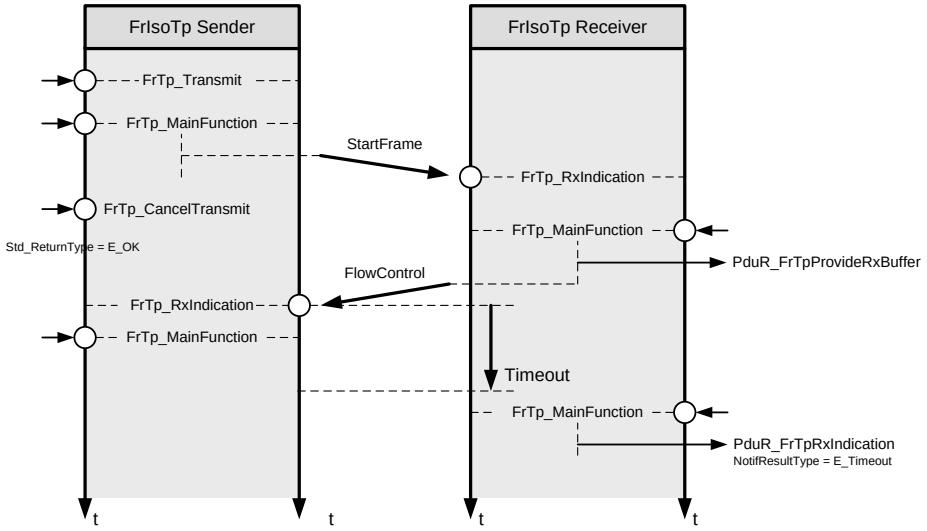


Figure 21: Transmit Cancellation at segmented data transfer

If an ongoing data transfer shall be cancelled, the service primitive *FrTp_CancelTransmit()* is called. *FrTp_CancelTransmit()* shall cancel the current data transfer process. On receiver side an initial data reception is recognized and processed (e.g. call of service primitive *PduR_FrTpProvideRxBuffer*, send FlowControl N-PDU etc.). If the sender cancels data transfer a timeout occurs on receiver side.

If no retry is configured, this timeout is used to cancel the current reception by calling the service primitive *PduR_FrTpRxIndication* with the corresponding notification result error code.

If retry is configured, the receiver sends an additional FlowControl[28]. After a configured amount of retries the final timeout is used to cancel the current reception by calling the service primitive *PduR_FrTpRxIndication* with the corresponding notification result error code.

7.5.7 Change FrIsoTp Parameter

FRISOTP242: The FrIsoTp module shall change the ISO10681-2 FlowControl PDU parameter(s) of BandwidthControl (BC)
a) FrIsoTpSCexp (please refer to ISO 10681-2)
b) FrIsoTpMaxNbrOfNPduPerCycle (please refer to ISO 10681-2)
during runtime if the corresponding API service primitive *FrTp_ChangeParameter* is called.

FRISOTP1179]:⌈ The layout of the BC parameter shall be identical to the

[28] Note: On sender site the additional FlowControl is received as an unexpected N-PDU and is ignored.

layout in the FC(CTS) frame: The FrTpMaxNbrOfNPduPerCycle shall be placed in bits 0..2, the FrTpSCexp in the bits 3…7. The upper byte of the parameter is not used.⌋()

FRISOTP1115: A change parameter request during an ongoing reception shall be terminated with return value of E_NOT_OK.

FRISOTP1156: The FrIsoTp module shall use the new BandwidthControl parameters for the corresponding connection if the change was successfully executed.

Note: Bandwidth Control is part of the runtime parameter set. For details please refer to chapter 7.3.3.3.

7.5.8 Timing parameter and timeout behaviour

The FrIsoTp module requires different timing parameters for communication handling. This chapter defines the timing and timeout behaviour.

FRISOTP1099: The FrIsoTp module shall support the different timers and their start/stop conditions for communication handling as defined in Figure 22, Figure 23 and Table 2.

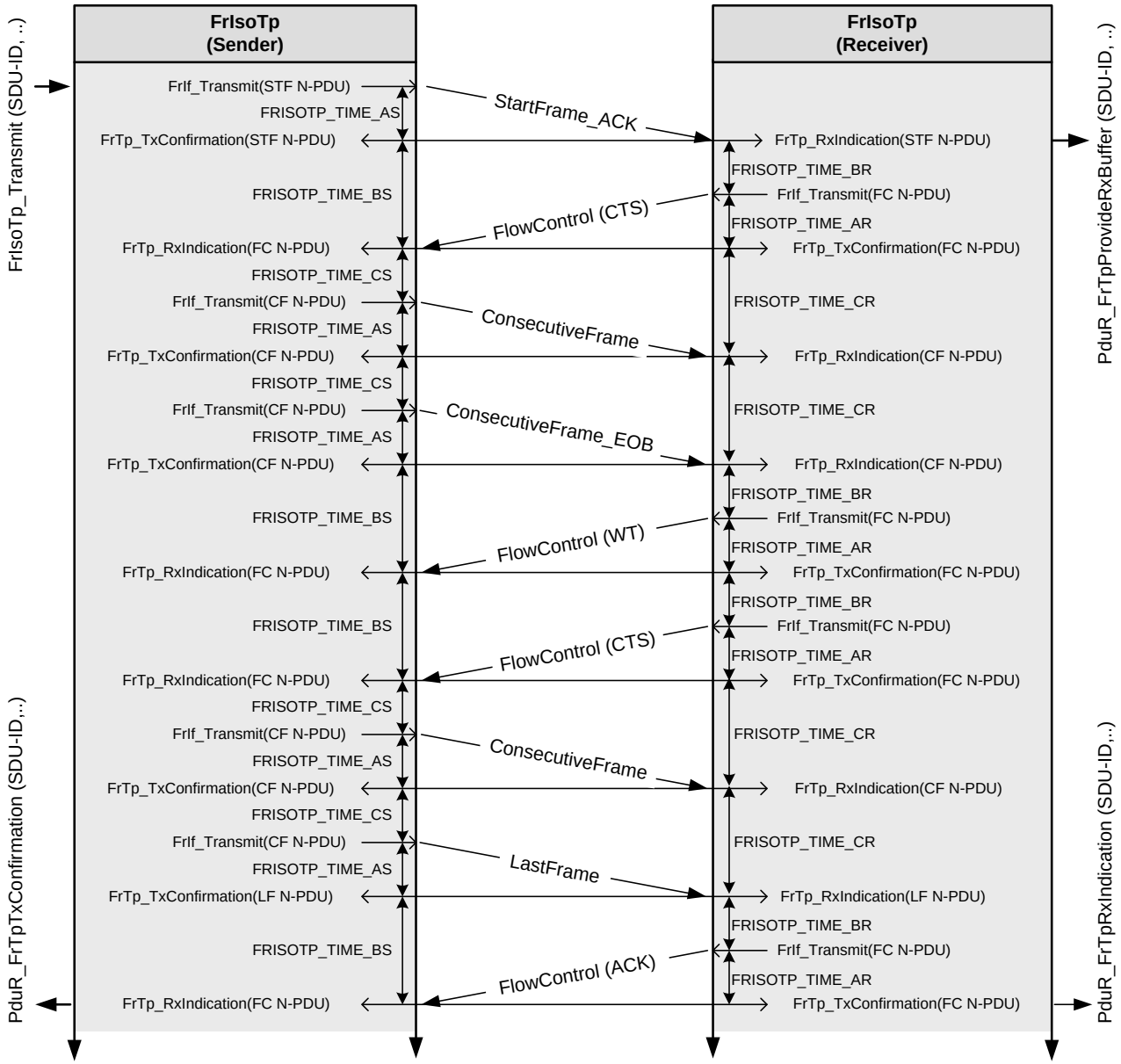


Figure 22: Timing parameter definition for one PDU transmission per Main-Function call

As described above it is possible to transmit more than one N-PDU per connection within on FlexRay Communication Cycle. Hence the communication handler shall be able to call FrIf_Transmit API several times (depending on available N-PDUs within the Tx-PDU-Pool) independent whether FrTp_TxConfirmation for the previously transmitted N-PDUs is given. Due to that, timing behavior (e.g. Start FRISOTP_Time_AS etc.) is different too. If more than one N-PDU shall be transmitted within one Main-Function call the timing behaviour is depict in Figure 23.

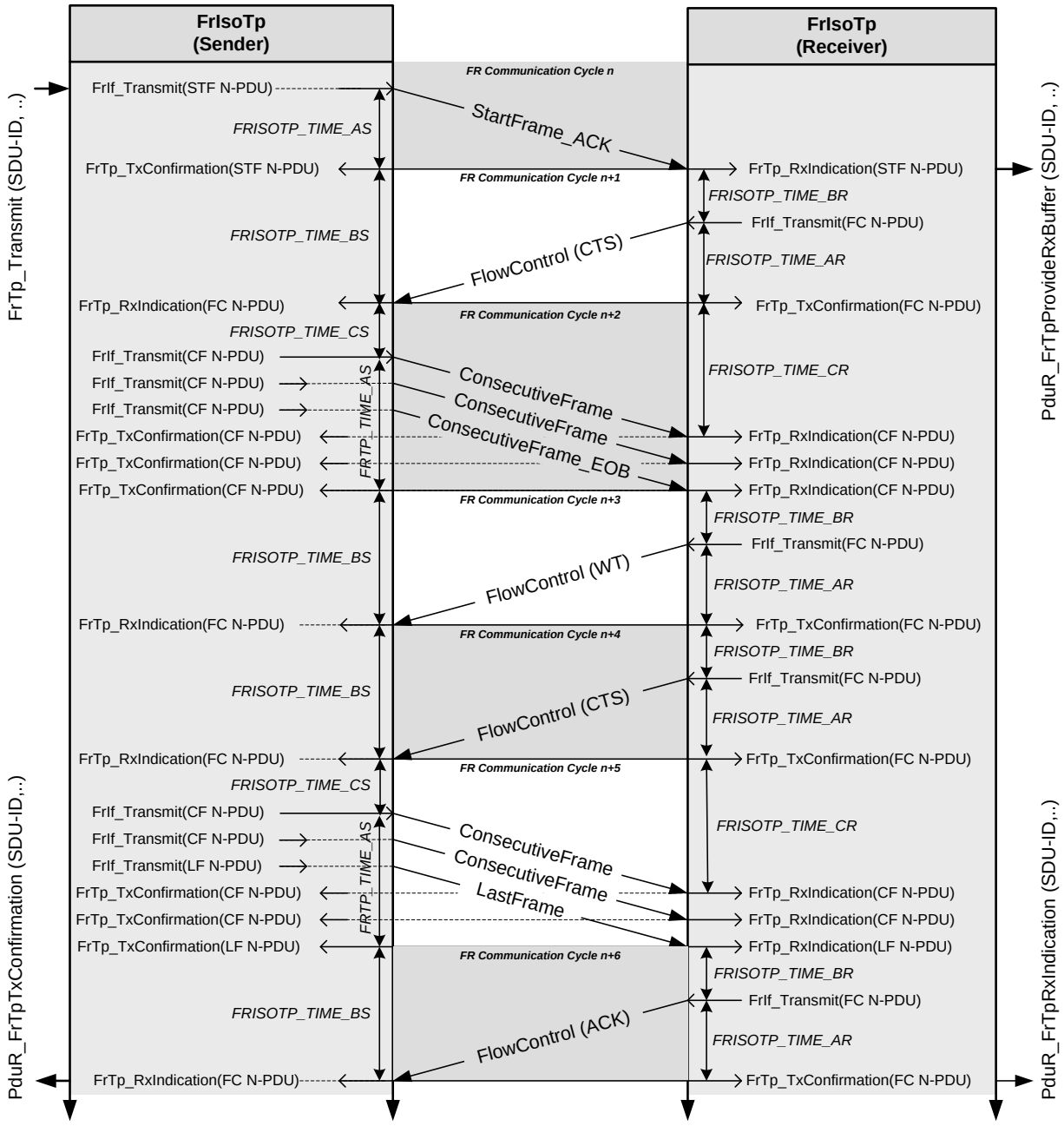


Figure 23: Timing parameter definition for multiple PDU transmission per Main-Function call

Note: Bandwidth Control restricts the number of N-PDUs per Flexray-Cycle. This has an impact to FrIsoTp_Time_CS and. Hence that time depends on implementation (task schedule of FrTp_MainFunction() and the corresponding FlowControl Parameters) For details please refer to chapter 7.3.3.3.

Timing Parameter	Description	Timer Start Condition	Timer Stop Condition
FRISOTP_TIME_AS	Time for transmission of any FrIsoTp N-PDU on the sender side. This is a performance requirement. The time depends on implementation and the global FlexRay schedule. The maximum time is controlled by the FRISOTP_TIMEOUT_AS.	FrIf_Transmit()	FrTp_TxConfirmation()
FRISOTP_TIME_AR	Time for transmission of FlowControl FrIsoTp N-PDU on the receiver side. This is a performance requirement. The time depends on implementation and the global FlexRay schedule. The maximum time is controlled by the FRISOTP_TIMEOUT_AR.	FrIf_Transmit()	FrTp_TxConfirmation
FRISOTP_TIME_BS	Time until reception of the next FlowControl N-PDU. The maximum time is controlled by the FRISOTP_TIMEOUT_BS.	FrTp_TxConfirmation (STF), FrTp_RxIndication (FC), FrTp_TxConfirmation (CF), FrTp_TxConfirmation (LF)	FrTp_RxIndication (FC)
FRISOTP_TIME_BR	Time until transmission of the next FlowControl N-PDU.	FrTp_RxIndication (STF), FrTp_TxConfirmation (FC), FrTp_RxIndication (CF), FrTp_RxIndication (LF)	FrIf_Transmit (FC)
FRISOTP_TIME_CR	Time until reception of the next ConsecutiveFrame N-PDU. The maximum time is controlled by the FRISOTP_TIMEOUT_CR.	FrTp_TxConfirmation (FC), FrTp_RxIndication (CF)	FrTp_RxIndication (CF) FrTp_RxIndication (LF)
S/s … sender R/r … receiver			

Table 2: Timing parameter for the FrIsoTp module

FRISOTP1100: The FrIsoTp module shall support the communication timeout behavior as defined in Table 3.

Timeout Parameter	Cause	Action
FRISOTP_TIMEOUT_AS	Any FrIsoTp N-PDU not transmitted in time on the sender side.[29]	Abort message transmission.[30] a) Free the FrIsoTpTxPdu. b) issue PduR_FrTpTxConfirmation with the corresponding FrIsoTpTxSduId and <NotifResultType> = NTFRSLT_E_TIMEOUT_A.
FRISOTP_TIMEOUT_AR	Any FrIsoTp FC N-PDU not transmitted in time on the receiver side.[31]	Issue PduR_RxIndication with the corresponding FrIsoTpTxSduId and <NotifResultType> = NTFRSLT_E_TIMEOUT_A.
FRISOTP_TIMEOUT_BS	FlowControl N-PDU not received (lost, overwritten) on the sender side.	Abort message transmission and issue PduR_FrTpTxConfirmation with the corresponding FrIsoTpTxSduId and <NotifResultType> = NTFRSLT_E_TIMEOUT_BS.
FRISOTP_TIMEOUT_CR	ConsecutiveFrame or Last Frame N-PDU not received (lost, overwritten) on the receiver side.[32]	Abort message reception and issue PduR_FrTpRxIndication with the corresponding FrIsoTpRxSduId and <NotifResultType> = NTFRSLT_E_TIMEOUT_CR.

Table 3: Timeout behaviour for FrIsoTp module

[29] This could occur if an N-PDU was suspended several times within the dynamic segment.

30 NOTE: In FlexRay the transmission confirmation doesn't provide an End-To-End confirmation as on other bus protocols (e.g. CAN). This means that a transmission confirmation is provided as soon/only if the L-PDU was passed over to the network. Hence, if no confirmation occurs, the L-PDU is still stuck within the message buffer of the FlexRay controller, which is occupied and cannot be used in the meanwhile.

[31] This could occur if an N-PDU is suspended several times within the dynamic segment.

32 This could occur in case preceding FlowControl N-PDU not received (lost, overwritten) on overall sender side.

7.6 Counters

Several counters are used to handle the different retry attempts. This chapter defines these different counters and their increasing and decreasing behaviour. Each counter is limited by a specified value. The figure below shows the different interactions between timer, counter and function calls.

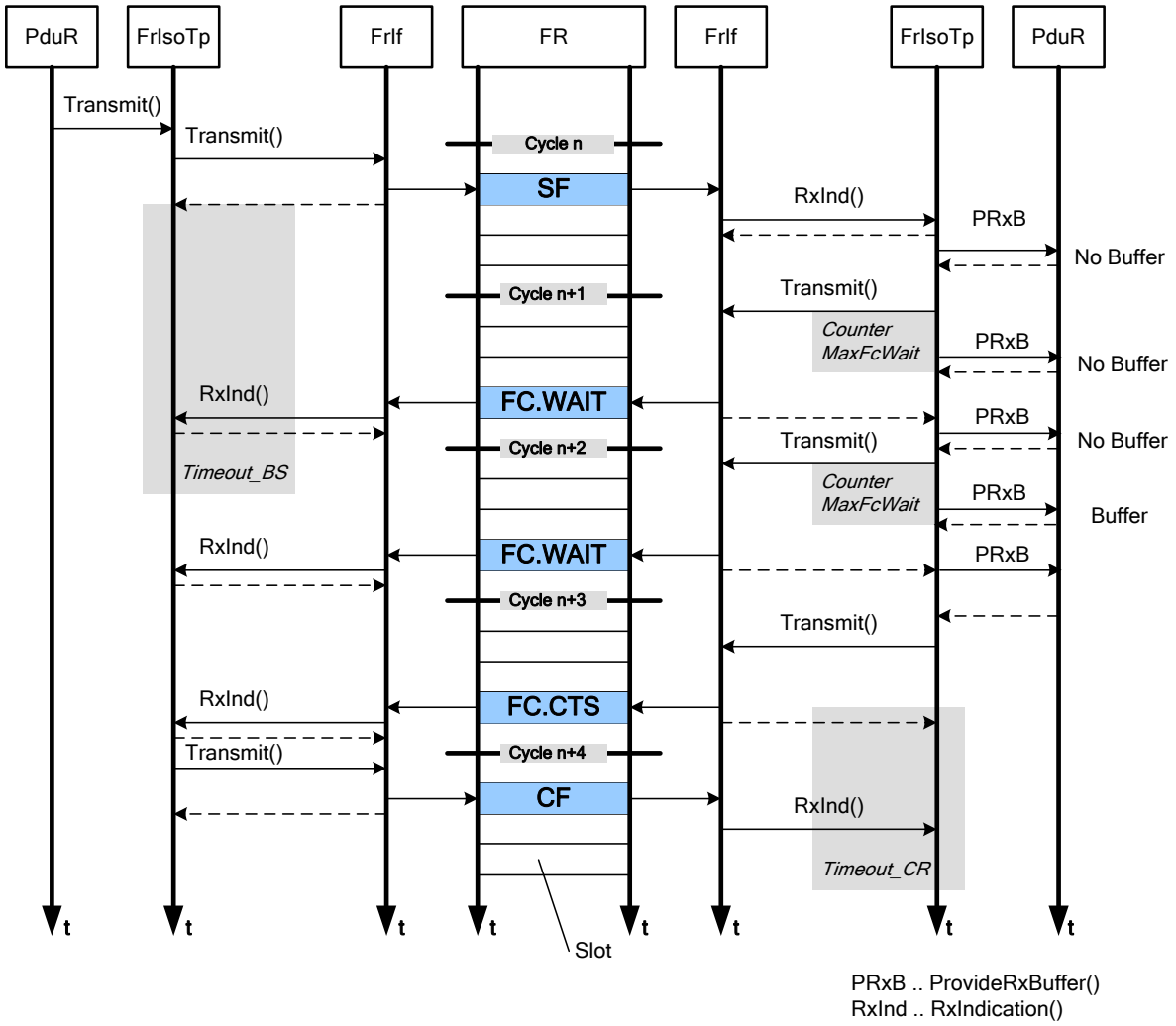


Figure 24: Counter, timer and function call interaction

FRISOTP1105: The FrIsoTp module shall support the counters and corresponding limits as defined in Table 4: FrIsoTp Counter.

FRISOTP1114: Each FrIsoTp Counter shall be limited by a max value as defines in Table 4.

Counter Name	Counter Description	Limit
COUNTER_FCWT	On receiver site: Counts the number of transmittes FlowControl.Wait frames[33]	FrIsoTpMaxFCWait
COUNTER_FRIF	Counts the attempts to send an Fr N-PDU via the service primitive *FrIf_Transmit*() in case this call returns E_NOT_OK	FrIsoTpMaxFrIf
COUNTER_AR	Counts the attempts of the receiver to send an Fr N-PDU (FC, AF), in case a timeout AR occurs	FrIsoTpMaxAr
COUNTER_AS	Counts the attempts to send an Fr N-PDU (SF, FF, LF) in case a timeout AS occurs	FrIsoTpMaxAs
Counter_RX_RN	Counts the transmission retry requests on receiver site initiated due to a frame error, e.g. bad SN in a CF.	FrIsoTpMaxRn

Table 4: FrIsoTp Counter

FRISOTP1113: If a counter of has been reached, the FrIsoTp module shall react as defined in Table 5.

Counter Name	Handling if counter has been reached
COUNTER_FCWT	Abort transmission
COUNTER_FRIF	Abort transmission
COUNTER_AR	Abort transmission
COUNTER_AS	Abort transmission
COUNTER_RX_RN	**Case a) Segmented – Acknowledged transmission** 1) Abort reception by calling service primitive PduR_FrTpRxIndication with NotifResultType = NTFRSLT_E_WRONG_SN 2) Send FlowControl.ABT (if aFlowControl is possible)

Table 5: FrIsoTp module reaction if counters reached

[33] The limit of COUNTER_FCWT shall be in relation to the task to get an RxBuffer (service primitive call PduR_FrTpProvideRxBuffer). The frequency of buffer request retries must be equal/higher than the FC.WAIT transmission frequency.

7.7 Error Handling

7.7.1 Error Detection

FRISOTP217: The detection of development errors shall be pre-compile time configurable by the configuration parameter: *FRISOTP_DEV_ERROR_DETECT*.

FRISOTP1163: The detection of development errors shall be (de)activated (*ON* / *OFF*) by the configuration parameter *FRISOTP_DEV_ERROR_DETECT*.

FRISOTP205: If the *FRISOTP_DEV_ERROR_DETECT* switch is enabled API parameter checking shall be enabled.

FRISOTP218: It shall be not possible to switch off production code errors detection.

Note: If no production error is currently specified the requirement FRISOTP218 could be disregarded.

FRISOTP1106: The *FrIsoTpState* shall be checked to detect whether FrIsoTp module is initialized or not.

7.7.2 Error Notification

FRISOTP206: Detected development errors shall be reported to the Development Error Tracer (DET) if the pre-processor switch *FrIsoTpDevErrorDetect* is set.

FRISOTP1110: Production errors shall be reported to Diagnostic Event Manager (DEM) [12].

Note: If no production error is currently specified the requirement FRISOTP1110 could be disregarded.

FRISOTP1107: The FrIsoTp module shall use the Development Error Tracer [13] service *Det_ReportError* to report development errors.

FRISOTP1108: The header file of the FrIsoTp module, `FrIsoTp.h`, shall provide a module Identifier `FRISOTP_MODULE_ID`.

FRISOTP1109: The module Identifier `FRISOTP_MODULE_ID` shall set to the value 0x24.

7.7.3 Error Classification

This section describes how the FrIsoTp module has to manage the several error classes that may occur during the life cycle of this basic software.

According to the general requirements on basic software modules [3] all basic software modules must distinguish (according to the product life cycle) two error types:

- Development errors:
  These errors should be detected and fixed during development phase. In most cases, these errors are software errors. The detection of errors that should only occur during development can be switched off for production code (by static configuration, namely preprocessor switches).

- Production errors:
  These errors are hardware errors and software exceptions that cannot be avoided and are expected to occur in the production (i.e. series) code.

FRISOTP1111: The FrIsoTp module shall support the error codes for development errors (Dev.) and production errors (Prod.) as defined in Table 6.

FRISOTP1112: Event Ids values for production code are assigned externally by the configuration of the DEM module. They are published in the file Dem_IntErrId.h and included via Dem.h.

FRISOTP179: Development error values are of type uint8.

FRISOTP1132: All errors which are listed within Table 6 and marked as "*Dev.*" in the column *Relevance* are classified as development errors.

Type of error	Relevance	Related error code	Value [hex]
API service call without module initialization: Exception: a) FrTp_Init() b) FrTp_GetVersionInfo()	Dev.	FRISOTP_E_UNINIT	0x01
NULL-Pointer on any API call	Dev.	FRISOTP_E_NULL_PTR	0x02
API call with invalid SDU-ID (PduR) or PDU-ID (FrIf)	Dev.	FRISOTP_E_INVALID_PDU_SDU_ID	0x03
API call with invalid Parameter	Dev.	FRISOTP_E_INVALID_PARAMETER	0x04
Segmentation is required for a 1:n connection	Dev.	FRISOTP_E_SEG_ERROR	0x05
Transmission of unknown message length is detected but not configured.	Dev.	FRISOTP_E_UMSG_LENGTH_ERROR	0x06
No free channel available[34]	Dev.	FRISOTP_E_NO_CHANNEL	0x07
Dev. .. Development			

Table 6: Module error classification

[34] No free channel could occure for each ECU in principle, but especially for gateway configuration this error shall indicate architecture/configuration problems.

8 API specification

8.1 Imported types

This chapter lists all included types for FlexRay ISO Transport Layer and their corresponding header files.

FRISOTP141: Std_ReturnType shall be imported from Std_Types.h.

FRISOTP1164: Std_VersionInfoType shall be imported from Std_Types.h.

FRISOTP1165: BufReq_ReturnType shall be imported from ComStack_Types.h.

FRISOTP1166: NotifResultType shall be imported from ComStack_Types.h.

FRISOTP1167: PduIdType shall be imported from ComStack_Types.h.

FRISOTP1168: PduInfoType shall be imported from ComStack_Types.h.

FRISOTP1169: PduLengthType shall be imported from ComStack_Types.h.

FRISOTP1178:TPParameterType shall be imported from ComStack_Types.h.

Module	*Imported Type*
ComStack_Types	BufReq_ReturnType
	NotifResultType
	PduIdType
	PduInfoType
	PduLengthType
	TPParameterType
Std_Types	Std_ReturnType
	Std_VersionInfoType

8.2 Type definitions

FRISOTP1133: The following FrIsoTp specific types shall be defined in FrIsoTp_Types.h.

8.2.1 AUTOSAR Notification Result Types (NotifResultType)

ISO10681-2 [16] and the FrIsoTp module are using several result types to store the result status of a notification (indication or confirmation). For the AUTOSAR Communication Stack a subset of notification result types is defined in specification [7]. This chapter specifies the mapping of ISO10681-2 notification result types to AUTOSAR notification result types.

8.2.1.1 General Return Codes

FRISOTP555: The FrIsoTp module shall support the AUTOSAR Notification Result Types as defined in specification [7].

FRISOTP557: The general AUTOSAR Notification Result Types as defined in specification [7] shall map to the ISO 10681-2 Notification Result Types as defined in Table 7.

Table 7: Mapping of AUTOSAR general notification result types to ISO-10681 result types

AUTOSAR - NotifResultCode	ISO10681-2 - C_Result	Value
NTFRSLT_E_ABORT	C_ABORT	Refer to specification [7]
NTFRSLT_E_INVALID_FS	C_INVALID_FS	Refer to specification [7]
NTFRSLT_E_NO_BUFFER	C_BUFFER_OVFLW	Refer to specification [7]
NTFRSLT_E_NOT_OK	C_ERROR	Refer to specification [7]
NTFRSLT_E_TIMEOUT_A	C_TIMEOUT_A	Refer to specification [7]
NTFRSLT_E_TIMEOUT_BS	C_TIMEOUT_Bs	Refer to specification [7]
NTFRSLT_E_TIMEOUT_CR	C_TIMEOUT_Cr	Refer to specification [7]
NTFRSLT_E_UNEXP_PDU	C_UNEXP_PDU	Refer to specification [7]
NTFRSLT_E_WFT_OVRN	C_WFT_OVRN	Refer to specification [7]
NTFRSLT_E_WRONG_SN	C_WRONG_SN	Refer to specification [7]
NTFRSLT_OK	C_OK	Refer to specification [7]

8.2.2 FrIsoTp_ConfigType

The post-build-time configuration fulfills two functionalities:

- Post-build selctable, where more than one configuration is located in the ECU, and one is selected at init of the FrIsoTp module
- Post-build loadable, where one configuration is located in the ECU. This configuration may be reprogrammed after compile-time

Basically there is no restriction to mix both selectable and loadable. Typically the post-build loadable is located in its own flash sector where it can be reprogrammed without affecting other modules/applications.

Name:	`FrIsoTp_ConfigType`
Type:	`Structure`
Range:	Implementation specific.
Description:	This is the base type for the configuration of the FlexRay ISO Transport Protocol A pointer to an instance of this structure will be used in the initialization of the FlexRay ISO Transport Protocol.

	The outline of the structure is defined in chapter 10 Configuration Specification

FRISOTP1137: The type FrIsoTp_ConfigType is an external data structure containing post-build-time configuration data of the FrIsoTp module which shall be implemented in FrIsoTp_PBcfg.c (see chapter 5.6.1).

8.3 Function definitions

8.3.1 Standard functions

8.3.1.1 FrTp_GetVersionInfo

FRISOTP215:

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.

FRISOTP202: The function FrTp_GetVersionInfo shall return the version information of this module. The version information includes:
- Two bytes for the vendor ID
- One byte for the module ID
- Three bytes version number.

The numbering shall be vendor specific; it consists of:
- The major, the minor and the patch version number of the module.

The AUTOSAR specification version number shall not be included. The AUTOSAR specification version number is checked during compile time and therefore not required in this API.

Note: Please refer also to document: AUTOSAR_SWS_StandardTypes.pdf .

FRISOTP498: The function FrTp_GetVersionInfo shall be pre compile time configurable On/Off by the configuration parameter: *FrIsoTpVersionInfoApi*

FRISOTP1150: If development error detection for the FrTp_GetVersionInfo is enabled: the function FrTp_GetVersionInfo shall check the parameter versioninfo for being valid. If the check for versioninfo fails, the function FrTp_GetVersionInfo shall raise the development error FRISOTP_E_NULL_PTR and return E_NOT_OK.

8.3.2 Initialization and Shutdown

8.3.2.1 FrTp_Init

FRISOTP147:

Service name:	FrTp_Init

Syntax:	`void FrTp_Init(`
	`  const FrIsoTp_ConfigType* configPtr`
	`)`

Service ID[hex]:	0x00
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	configPtr — Pointer to FlexRay ISO Transport Protocol configuration.
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This service initializes all global variables of a FlexRay ISO 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).

FRISOTP1151: If development error detection for the FrTp_Init is enabled: the function FrTp_Init shall check the parameter configPtr for being valid. If the check for configPtr fails, the function FrTp_Init shall raise the development error FRISOTP_E_NULL_PTR and return E_NOT_OK.

8.3.2.2 FrTp_Shutdown

FRISOTP148:

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 FrIsoTp Module

8.3.3 Normal Operation

8.3.3.1 FrTp_Transmit

FRISOTP149:

.

Service name:	FrTp_Transmit	
Syntax:	`Std_ReturnType FrTp_Transmit( PduIdType FrIsoTpTxSduId, const PduInfoType* FrIsoTpTxSduInfoPtr )`	
Service ID[hex]:	0x02	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant	
Parameters (in):	FrIsoTpTxSduId	This parameter contains the FlexRay TP instance unique identifier of the FrIsoTp N-SDU to be transmitted.
	FrIsoTpTxSduInfoPtr	Tx N-SDU Information Structure which contains a) pointer to the FrIsoTp Tx N-SDU b) the length of the FrIsoTp Tx N-SDU
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. parameter check has failed or no FrIsoTpChannel resource is free.
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 FrIsoTp's SDU-Id, i.e. the upper layer has to translate its own PDU-Id into the FrIsoTp's SDU-ID for the corresponding message. Within the provided FrIsoTpSduInfoPtr 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.	

Note: The service primitive FrTp_Transmit sets the flag TX_SDU_AVAILABLE if new data are available for transmission.

FRISOTP1139: If development error detection for the FrTp_Transmit is enabled: the function FrTp_Transmit shall check the parameter FrIsoTpTxSduId for being valid. If the check for FrIsoTpTxSduId fails, the function FrTp_Transmit shall raise the development error FRISOTP_E_INVALID_PDU_SDU_ID and return E_NOT_OK.

FRISOTP1140: If development error detection for the FrTp_Transmit is enabled: the function FrTp_Transmit shall check the parameter FrIsoTpTxSduInfoPtr for being valid. If the check for FrIsoTpTxSduInfoPtr fails, the function FrTp_Transmit shall raise the development error FRISOTP_E_NULL_PTR and return E_NOT_OK.

8.3.3.2 FrTp_CancelTransmit

FRISOTP150:

Service name:	FrTp_CancelTransmit	

Syntax:	`Std_ReturnType FrTp_CancelTransmit(`
	`  PduIdType FrIsoTpTxPduId`
	`)`	

Service ID[hex]:	0x03	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	FrIsoTpTxPduId	This parameter contains the FlexRay ISO TP instance unique identifier of the Fr N-SDU which transfer has to be cancelled.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Transmit cancellation request of the specified Fr N-SDU is accepted E_NOT_OK: Transmit cancellation request of the specified Fr N-SDU is rejected for an unsegmented data transmission
Description:	This service primitive is used to cancel the transfer of pending Fr N-SDUs. The connection is identified by FrIsoTpTxSduId. When the function returns, no transmission is in progress anymore with the given N-SDU identifier. This function has to be called with the PDU-Id of the FrIsoTp, i.e. the upper layer has the same PDU-Id as for the FrTp_Transmit() call.	

FRISOTP1141: If development error detection for the FrTp_CancelTransmit is enabled: the function FrTp_CancelTransmit shall check the parameter FrIsoTpTxPduId for being valid. If the check for FrIsoTpTxPduId fails, the function FrTp_CancelTransmit shall raise the development error FRISOTP_E_INVALID_PDU_SDU_ID and return E_NOT_OK.

8.3.3.3 FrTp_ChangeParameter:

FRISOTP151:

Service name:	FrTp_ChangeParameter	

Syntax:	`Std_ReturnType FrTp_ChangeParameter(`
	`  PduIdType id,`
	`  TPParameterType parameter,`
	`  uint16 value`
	`)`	

Service ID[hex]:	0x04	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant	
Parameters (in):	id	Identification of the I-PDU to which the parameter the request shall affect.
	parameter	The selected parameter that the request shall change. Only the parameter TP_BC is accepted by FrIsoTp.
	value	The value that the request shall change to. Range: $0000 - $00FF

Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: request is accepted E_NOT_OK: request is not accepted
Description:	Request to change transport protocol parameter BandwithControl.	

FRISOTP1143: If development error detection for the FrTp_ChangeParameter is enabled: the function FrTp_ChangeParameter shall check the parameter Id for being valid. If the check for Id fails, the function FrTp_ChangeParameter shall raise the development error FRISOTP_E_INVALID_PDU_SDU_ID and return E_NOT_OK.

FRISOTP1144: If development error detection for the FrTp_ChangeParameter is enabled: the function FrTp_ChangeParameter shall check the parameter for being valid. If the check for parameter fails, the function FrTp_ChangeParameter shall raise the development error FRISOTP_E_INVALID_PARAMETER and return E_NOT_OK.

8.3.3.4 FrTp_CancelReceive

FRISOTP1172:

Service name:	FrTp_CancelReceive	

Syntax:	`Std_ReturnType FrTp_CancelReceive(`
	`  PduIdType FrIsoTpRxSduId`
	`)`	

Service ID[hex]:	0x08	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	FrIsoTpRxSduId	SDU-Id of currently ongoing reception
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Reception was terminated sucessfully E_NOT_OK: Reception was not terminated
Description:	By calling this API with the corresponding RxSduId, the currently ongoing data reception is terminated immediately. When the function returns, no reception is in progress anymore with the given N-SDU identifier.	

8.4 Call-back notifications

8.4.1 FrTp_TriggerTransmit

FRISOTP154:

Service name:	FrTp_TriggerTransmit	
Syntax:	`Std_ReturnType FrTp_TriggerTransmit( PduIdType FrIsoTpTxPduId, PduInfoType* FrIsoTpTxPduInfoPtr )`	
Service ID[hex]:	0x07	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	FrIsoTpTxPduId	This parameter contains the FlexRay TP instance unique identifier of the FrIsoTp N-SDU to be transmitted.
Parameters (inout):	None	
Parameters (out):	FrIsoTpTxPduInfoPtr	Tx N-PDU Information Structure which contains a) pointer to the FrIsoTp Tx N-PDU b) the length of the FrIsoTp Tx N-PDU
Return value:	Std_ReturnType	E_OK: The request has been accepted E_NOT_OK: The request has not been accepted, e. g. parameter check has failed or the call does not occur in the context of active session
Description:	This service primitive is called by the FlexRay Interface to get the payload data (FrIsoTp N-PDU) and the payload data length for the transmit request of the FrIsoTp. The trigger transmit is initiated by the FlexRay schedule. This function has to be called with the PDU-Id of the FrIsoTp, i.e. the upper layer has to translate the upper layer's PDU-Id into FrIsoTp's SDU-Id for the corresponding message.	

FRISOTP1145: If development error detection for the FrTp_TriggerTransmit is enabled: the function FrTp_TriggerTransmit shall check the parameter FrIsoTpTxPduId for being valid. If the check for FrIsoTpTxPduId fails, the function FrTp_TriggerTransmit shall raise the development error FRISOTP_E_INVALID_PDU_SDU_ID and return E_NOT_OK.

FRISOTP1146: If development error detection for the FrTp_TriggerTransmit is enabled: the function FrTp_TriggerTransmit shall check the parameter FrIsoTpTxPduInfoPtr for being valid. If the check for FrIsoTpTxPduInfoPtr fails, the function FrTp_TriggerTransmit shall raise the development error FRISOTP_E_NULL_PTR and return E_NOT_OK.

8.4.2 FrTp_RxIndication

FRISOTP152:

Service name:	FrTp_RxIndication	

Syntax:	`void FrTp_RxIndication(`
	`  PduIdType FrIsoTpRxPduId,`
	`  const PduInfoType* FrIsoTpRxPduInfoPtr`
	`)`	

Service ID[hex]:	0x05	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	FrIsoTpRxPduId	This parameter contains the identifier of the received FrIsoTp N-PDU.
	FrIsoTpRxPduInfoPtr	Rx N-PDU Information Structure which contains a) pointer to the received FrIsoTp Rx N-PDU b) the length of the received FrIsoTp Rx N-PDU
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	The FlexRay Interface calls this primitive after reception of an Fr N-PDU. Within this routine the FlexRay ISO Transport Layer module copies the received PDU to a local buffer. This routine shall be called with the (Rx) PDU-ID of the FrIsoTp, i.e. the FlexRay Interface has to translate its own PDU-ID into the corresponding one of the FrIsoTp.	

FRISOTP1147: If development error detection for the FrTp_RxIndication is enabled: the function FrTp_RxIndication shall check the parameter FrIsoTpRxPduId for being valid. If the check for FrIsoTpRxPduId fails, the function FrTp_RxIndication shall raise the development error FRISOTP_E_INVALID_PDU_SDU_ID and return E_NOT_OK.

FRISOTP1148: If development error detection for the FrTp_RxIndication is enabled: the function FrTp_RxIndication shall check the parameter FrIsoTpRxPduInfoPtr for being valid. If the check for FrIsoTpRxPduInfoPtr fails, the function FrIsoTpRxPduInfoPtr shall raise the development error FRISOTP_E_NULL_PTR and return E_NOT_OK.

8.4.3 FrTp_TxConfirmation

FRISOTP153:

Service name:	FrTp_TxConfirmation	

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

Service ID[hex]:	0x06	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	FrIsoTxPduId	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 ISO TP shall route this confirmation to the configured target transport connection. All transmitted FlexRay frames belonging to the FlexRay ISO TP shall be confirmed by using this function. This function has to be called with the PDU-Id of the FrIsoTp, i.e. the FlexRay Interface has to translate its own PDU-Id into the corresponding one of the FrIsoTp.	

FRISOTP1149: If development error detection for the FrTp_TxConfirmation is enabled: the function FrTp_TxConfirmation shall check the parameter FrIsoTxPduId for being valid. If the check for FrTxPduId fails, the function FrTp_TxConfirmation shall raise the development error FRISOTP_E_INVALID_PDU_SDU_ID and return E_NOT_OK.

8.5 Scheduled functions

Basic Software Scheduler directly calls these functions.

8.5.1 FrTp_MainFunction

FRISOTP203: The `FrTp_MainFunction` shall be used to schedule the FrIsoTp module.

FRISOTP580: The `FrTp_MainFunction` shall be the entry point for FrIsoTp processing tasks.

FRISOTP1152: The `FrTp_MainFunction` shall be called at least one time per FlexRay cycle.[35]

FRISOTP162: The `FrTp_MainFunction` shall follow the service primitive definition as described below:

Service name:	FrTp_MainFunction

Syntax:	`void FrTp_MainFunction(`
	
	`)`

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

Timing parameter 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 could be defined. The function is if conditions are fulfilled. Alternatively, the function may be called cyclically however the cycle time will be assigned dynamically during runtime by other modules.

[35] The number of MainFunction calls depends on the global Flexray communication cycle length, the available reveive buffers of the FlexRay driver and the implementation (which functionality of transmission and reception could be implemented in interrupt mode). At least one call is necessary to reconfigure the buffers for the corresponding cycle.
If more than one call is necessary it is recommended to call MainFunction at the start of the static segment and at the start of the dynamic segment within the communication cycle. If the length of that segments are asymetic the different segment lengths have to be considered.

8.6 Expected Interfaces

This chapter describes all expected APIs from other modules.

8.6.1 Mandatory Interfaces

This chapter defines all mandatory interfaces (API service primitives), which are required in order t fulfill the core functionality of the FrIsoTp module.

8.6.1.1 PDU Router Interface

FRISOTP577: The FrIsoTp module expects service primitives from the PDU Router as listed in Table 8.

API service primitive	Description
PduR_FrTpRxIndication	By this API service primitive, the FrIsoTp indicates the completed (un)successful reception of a message.
PduR_FrTpProvideRxBuffer	By this API service primitive, the FrIsoTp module indicates that the actual received N-SDU data shall be delivered to the receiver module (e.g. COM, DCM etc.).
PduR_FrTpProvideTxBuffer	By this API service primitive, the FrIsoTp module requests the actual sender module (e.g. COM, DCM etc.) of the Fr N-SDU to provide a transmit data.
PduR_FrTpTxConfirmation	By this API service primitive, the FrIsoTp module confirms the (un)successful sending of the complete Fr N-SDU to the corresponding sender module (e.g. COM, DCM etc).

Table 8: Required PDU Router service primitives

8.6.1.2 FlexRay Interface

FRISOTP578: The FrIsoTp module expects service primitives from the FlexRay Interface as listed in Table 9.

API service primitive	Description
FrIf_Transmit	By this API service primitive, the FrIsoTp module initiates a transmission of an N-PDU.

Table 9: Required FlexRay Interface service primitives

8.6.2 Optional Interfaces

8.6.2.1 Debug Error Tracer Interface

FRISOTP579: Depending on the configuration parameter `FrIsoTpDevErrorDetect`, the FrIsoTp module expects service primitives from Debug Error Tracer module as listed in Table 10.

API service primitive	Description
Det_ReportError	By this API service primitive, development errors are reported.

Table 10: Required Debug Error Tracer service primitives

8.6.3 Configurable interfaces

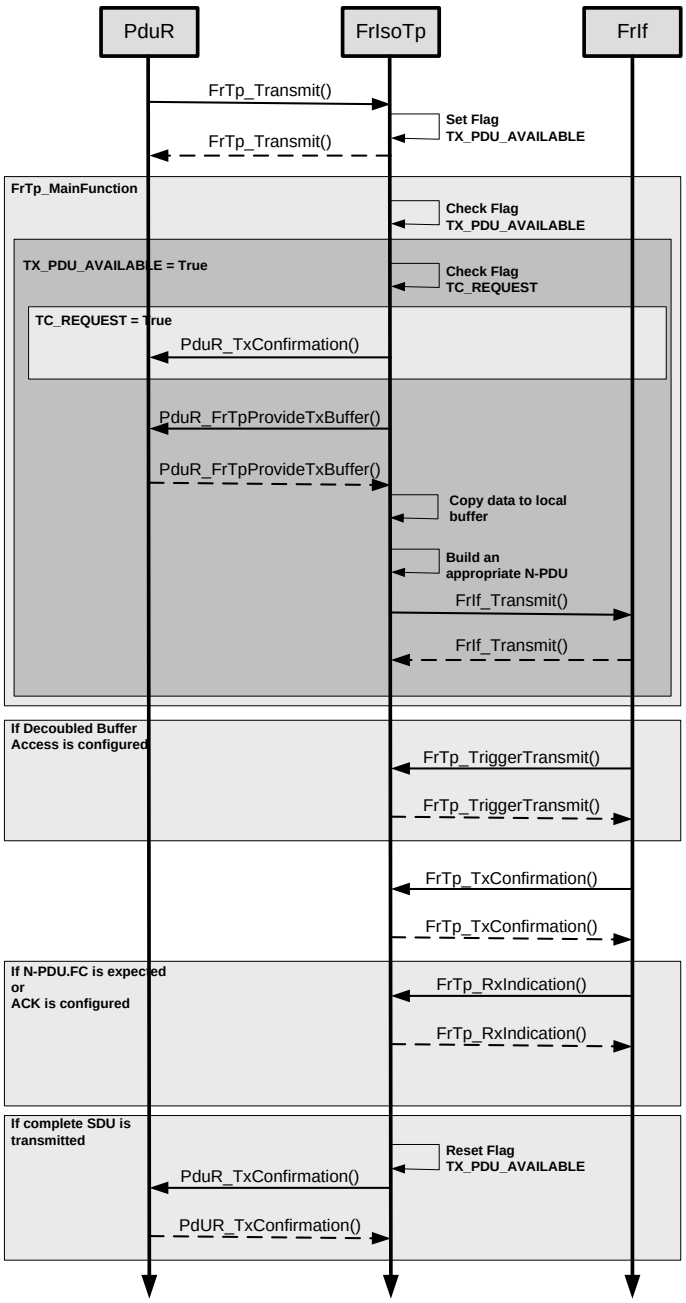
No interfaces are defined.

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.

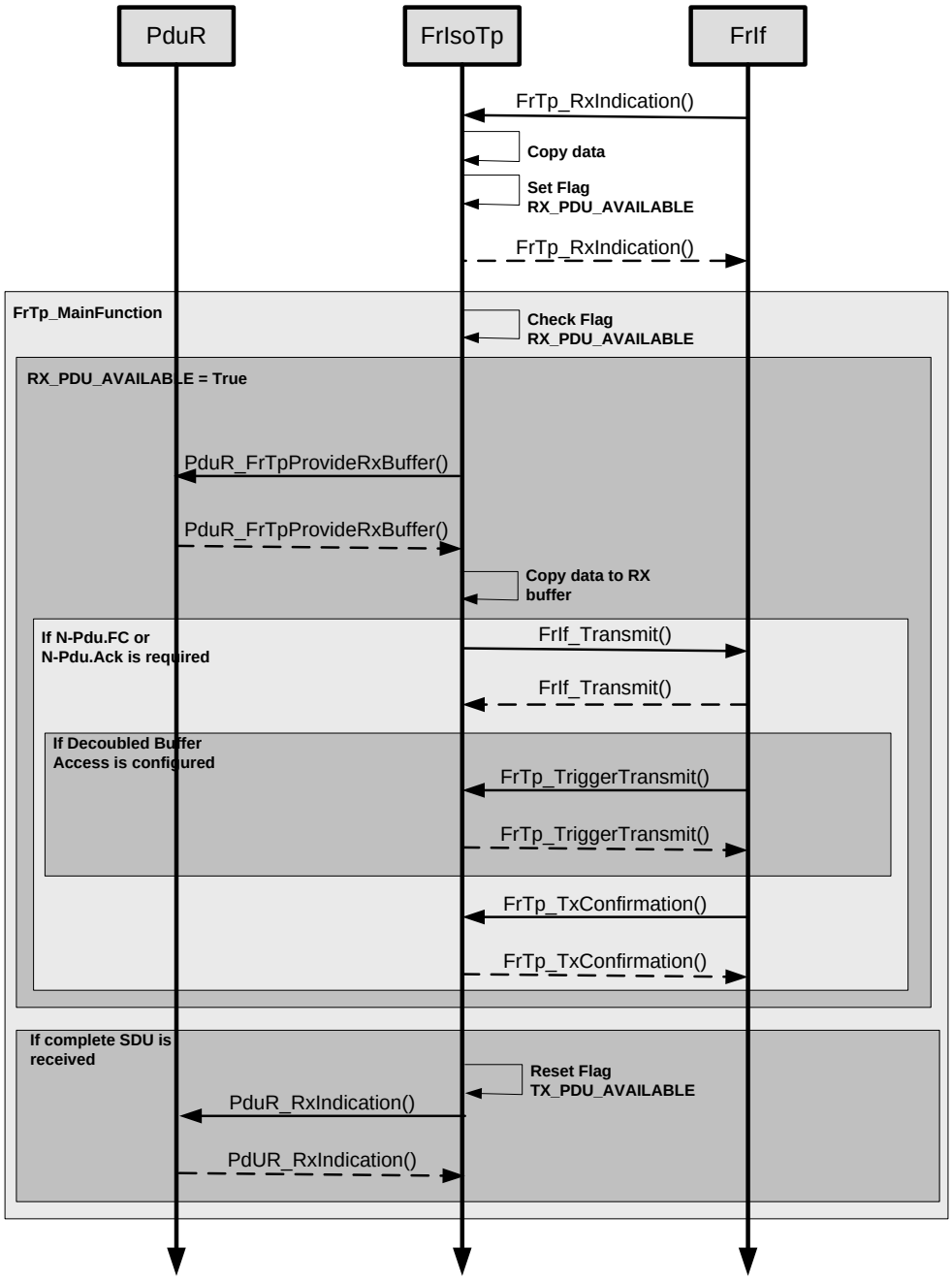
9.1 Sending of N-Pdus

The flow chart below depicts the sending process' API calls and flag handling. The flow chart shows segmented/unsegmented transfer, Transfer with and without acknowledgement and immediate / decoubled buffer access.



9.2 Receiving of N-Pdus

The flow chart below depicts the receiving process' API calls and flag handling. The flow chart shows segmented/unsegmented transfer, Transfer with and without acknowledgement and immediate / decoubled buffer access.



10 Configuration specification

This chapter defines configuration parameters and their clustering into containers. In order to support the specification, Chapter 10.1 describes fundamentals.

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

Chapter 10.3 specifies published information for the module FlexRay ISO Transport Layer module.

FRISOTP569: The configuration tool should extract all information to configure the FlexRay ISO Transport Protocol.

FRISOTP570: The configuration tool shall check the configuration consistency at configuration time. Configuration rules and constraints for plausibility checks shall performed where ever possible, during configuration time.

10.1 How to read this chapter

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

- AUTOSAR Layered Software Architecture [2].
  This document describes the AUTOSAR layered software architecture and shows the involved software modules.

- AUTOSAR Specification of Communication Stack Types [7].
  This document describes the basic data types for the communication stack. This data types are used in the configuration parameter containers.

- AUTOSAR ECU Configuration Specification [11].
  This document describes the AUTOSAR configuration methodology and the AUTOSAR configuration meta model 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 module implementation. 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 within the software generation process:

- before compile time,
- before link time,
- after build time.

In the following, the term “configuration class” (of a parameter) shall be used in order to refer to a specific configuration point in time.

10.1.2 Variants

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

10.1.3 Containers

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 configuration parameter containers consist of three sections:

- the general section
  includes general information of a container
- the configuration parameter section
  includes the configuration information of the specific parameter of a container
- the section of included containers
  describes the sub-containers which are also part of that (parent) container.

Table 11 depicts a detailed description of the different container items.

General Section			
SWS Item			
Container Name	Identifies the container by name		
Description	Explains the intention and content of the container.		
Configuration Parameters			
Name	Identifies the parameter by name.		
Description	Explains the intention of the configuration parameter.		
Type or Unit	Specifies the type of parameter (e.g., `uint8..uint32`) or specifies the unit of the parameter (e.g., `ms`)		
Range	Specifies the range (or possible values) of the parameter (e.g., `1..15`, `ON,OFF`)		Description(s) of the value(s) or range(s).
Configuration Class	***Pre-compile***	See description below.	Refer here to (a) variant(s).
	Link time	See description below.	Refer here to (a) variant(s).
	Post Build	See description below.	Refer here to (a) variant(s).
Scope	The scope describes the impact of the configuration parameter: Does the setting affect only one instance of the module (instance), all instances of this module (module), the ECU or a network? *Possible values of scope: instance, module, ECU, network*		
Dependency	*Describes the dependencies to other parameters, containers or global configurations.*		

Included Container(s)		
Container Name	***Multiplicity***	***Scope / Dependency***
Reference a valid (sub)container by its name.	Specifies the number of possible instances of the referenced container.	The scope describes the impact of the configuration parameter: Does the setting affect only one instance of the module (instance), all instances of this module (module), the ECU or a network?

Table 11: Description of the container items

Pre-compile time:
This item 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 for the FrIsoTp module.

10.2.1 Variants

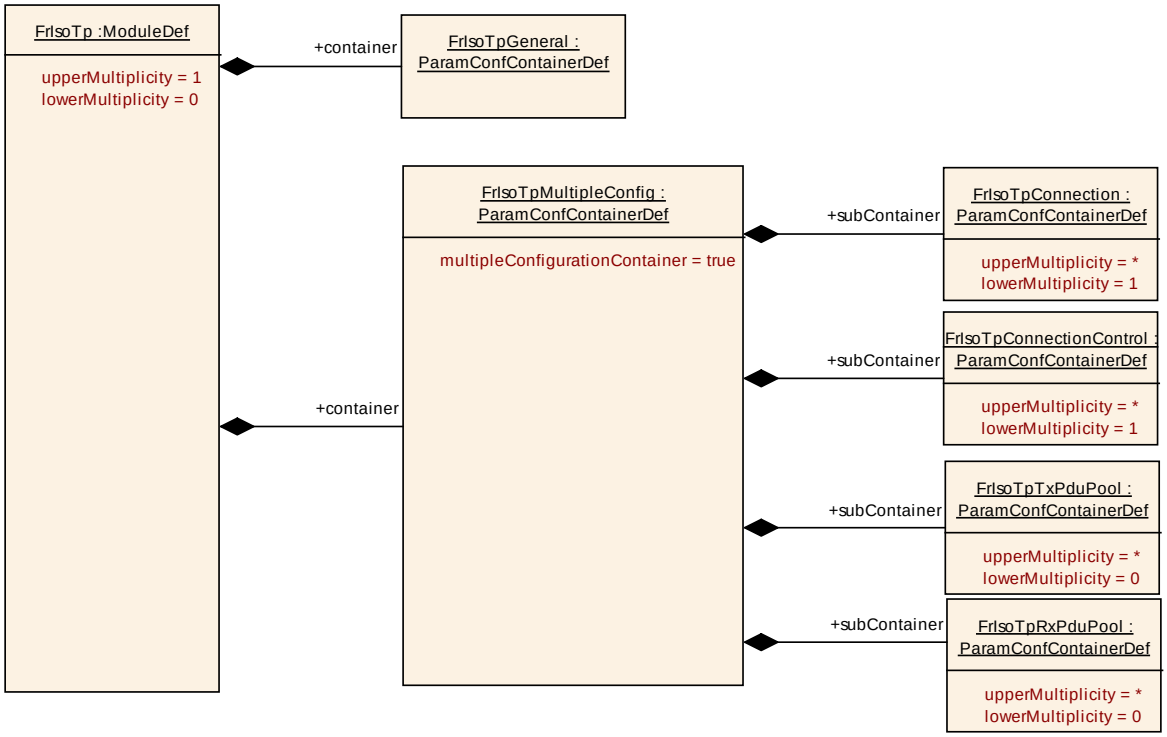
FRISOTP1131: The FrIsoTp module shall support configuration variants as listed below:

VARIANT-PRE-COMPILE	Only parameters with "Pre-compile time" configuration are allowed in this variant.
VARIANT-POST-BUILD	Parameters with "Pre-compile time" and "Post-build time" are allowed in this variant.

10.2.2 FrIsoTp

SWS Item	**FRISOTP001_Conf :**
Module Name	*FrIsoTp*
Module Description	Configuration of the FlexRay ISO Transport Protocol module according to ISO 10681-2.

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
FrIsoTpGeneral	1	This container contains the general configuration parameters of the FlexRay ISO Transport Protocol module.
FrIsoTpMultipleConfig	1	This container holds one or several multiple configuration sets.



10.2.3 FrIsoTpGeneral

SWS Item	**FRISOTP009_Conf :**
Container Name	FrIsoTpGeneral
Description	This container contains the general configuration parameters of the FlexRay ISO Transport Protocol module.
Configuration Parameters	

SWS Item	**FRISOTP002_Conf :**		
Name	FrIsoTpAckRt {FRISOTP_HAVE_ACKRT}		
Description	Preprocessor switch for enabling the Acknowledgement and retry mechanisms. True: Acknowledge and Retry is enabled False: Acknowledge and Retry is disabled		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**FRISOTP004_Conf :**		
Name	FrIsoTpChanNum {FRISOTP_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	**FRISOTP008_Conf :**		
Name	FrIsoTpDevErrorDetect {FRISOTP_DEV_ERROR_DETECT}		
Description	Preprocessor switch for enabling development error detection. True: Development Error Detection is enabled False: Development Error Detection is disabled		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**FRISOTP051_Conf :**		
Name	FrIsoTpFullDuplexEnable {FRISOTP_FULL_DUPLEX_ENABLE}		
Description	Preprocessor switch for enabling full duplex mechanisms for all channels. True: Full duplex is enabled False: Fullduplex is disabled (Half duplex is enabled)		
Multiplicity	1		
Type	BooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**FRISOTP011_Conf :**		
Name	FrIsoTpMainFuncCycle {FRISOTP_MAINFUNC_CYCLE}		
Description	This parameter contains the calling period of the TPs Main Function. The parameter is specified in seconds.		
Multiplicity	1		
Type	FloatParamDef		
Range	-INF .. INF		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

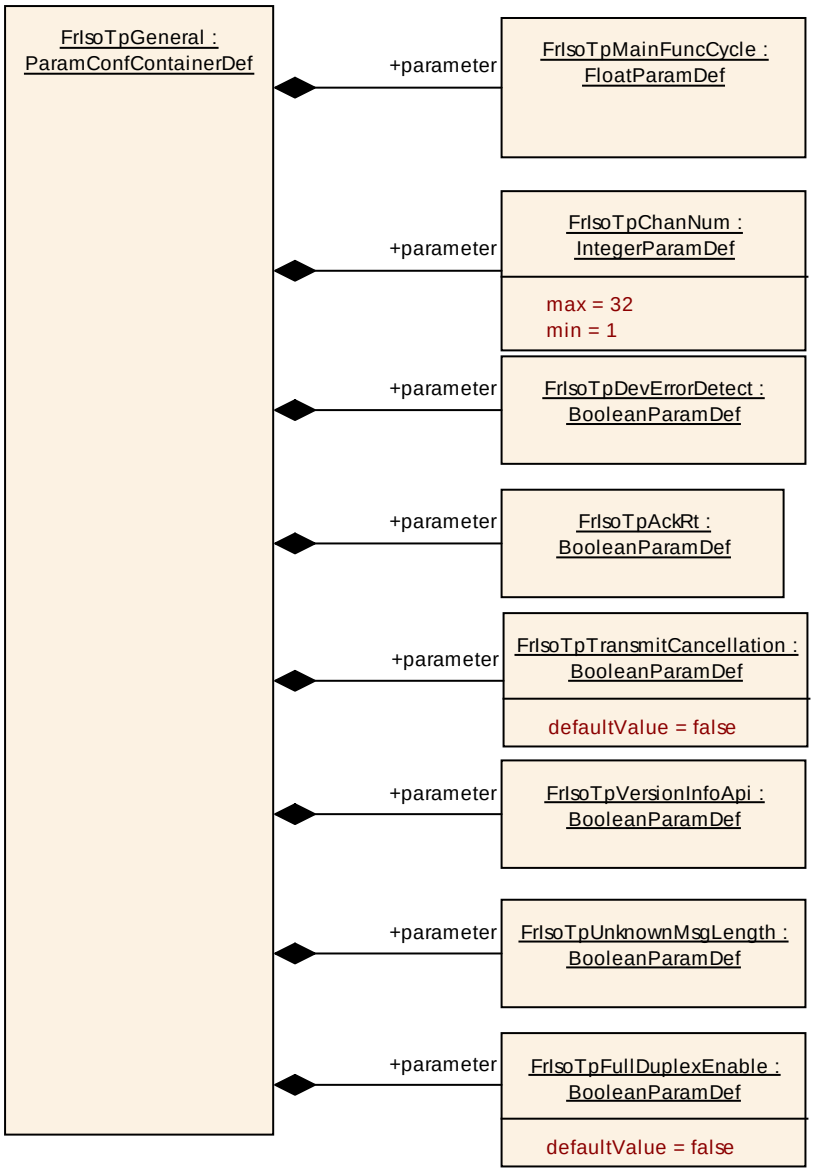
SWS Item	**FRISOTP036_Conf :**		
Name	FrIsoTpTransmitCancellation {FRISOTP_HAVE_TC}		
Description	Preprocessor switch for enabling Transmit Cancellation and Receive Cancellation. True: Transmit/Receive Cancellation is enabled False: Transmit/Receive Cancellation is disabled		
Multiplicity	1		
Type	BooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	All Variants

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

SWS Item	**FRISOTP044_Conf :**		
Name	FrIsoTpUnknownMsgLength		
Description	Preprocessor switch to support data transfer with unknown message length. True: Transmission with unknown message length is enabled False: Transmission with unknown message length is disabled		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**FRISOTP045_Conf :**		
Name	FrIsoTpVersionInfoApi {FRISOTP_VERSION_INFO_API}		
Description	Preprocessor switch for enabling the Version info API. True: Version Info API is enabled False: Version Info API is disabled		
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



10.2.4 FrIsoTpMultipleConfig

SWS Item	**FRISOTP018_Conf :**
Container Name	FrIsoTpMultipleConfig [Multi Config Container]
Description	This container holds one or several multiple configuration sets.
Configuration Parameters	

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
FrIsoTpConnection	1..*	This container contains the connection specific parameters to transfer N-PDUs via FlexRay ISO TP.
FrIsoTpConnectionControl	1..*	This container contains the configuration parameters to control a FlexRay ISO TP connection.
FrIsoTpRxPduPool	0..*	This container contains all Pdus that are assigned to that Pdu Pool.
FrIsoTpTxPduPool	0..*	This container contains all Pdus that are assigned to that Pdu Pool.

10.2.5 FrIsoTpConnection

SWS Item	**FRISOTP006_Conf :**
Container Name	FrIsoTpConnection{FrIsoTpConnectionConfiguration}
Description	This container contains the connection specific parameters to transfer N-PDUs via FlexRay ISO TP.
Configuration Parameters	

SWS Item	**FRISOTP050_Conf :**		
Name	FrIsoTpBandwidthLimitation {FRISOTP_BW_LIMIT}		
Description	This parameter indicates wheather the connection requires a bandwidth limitation or not. If FrIsoTpBandwidthLimitation=True the sender shall send a StartFrame always on the first PDU of a PDU-Pool.		
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	**FRISOTP010_Conf :**		
Name	FrIsoTpLa {FRISOTP_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.		
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	**FRISOTP019_Conf :**		
Name	FrIsoTpMultipleReceiverCon {FRISOTP_MULT_REC}		
Description	This parameter defines, whether this connection is an 1:1 ('false') or an 1:n ('true') connection. If this parameter is set to true the size of associated N-SDUs shall not be larger than the size of the associated N-PDUs.		
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	**FRISOTP021_Conf :**
Name	FrIsoTpRa {FRISOTP_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.

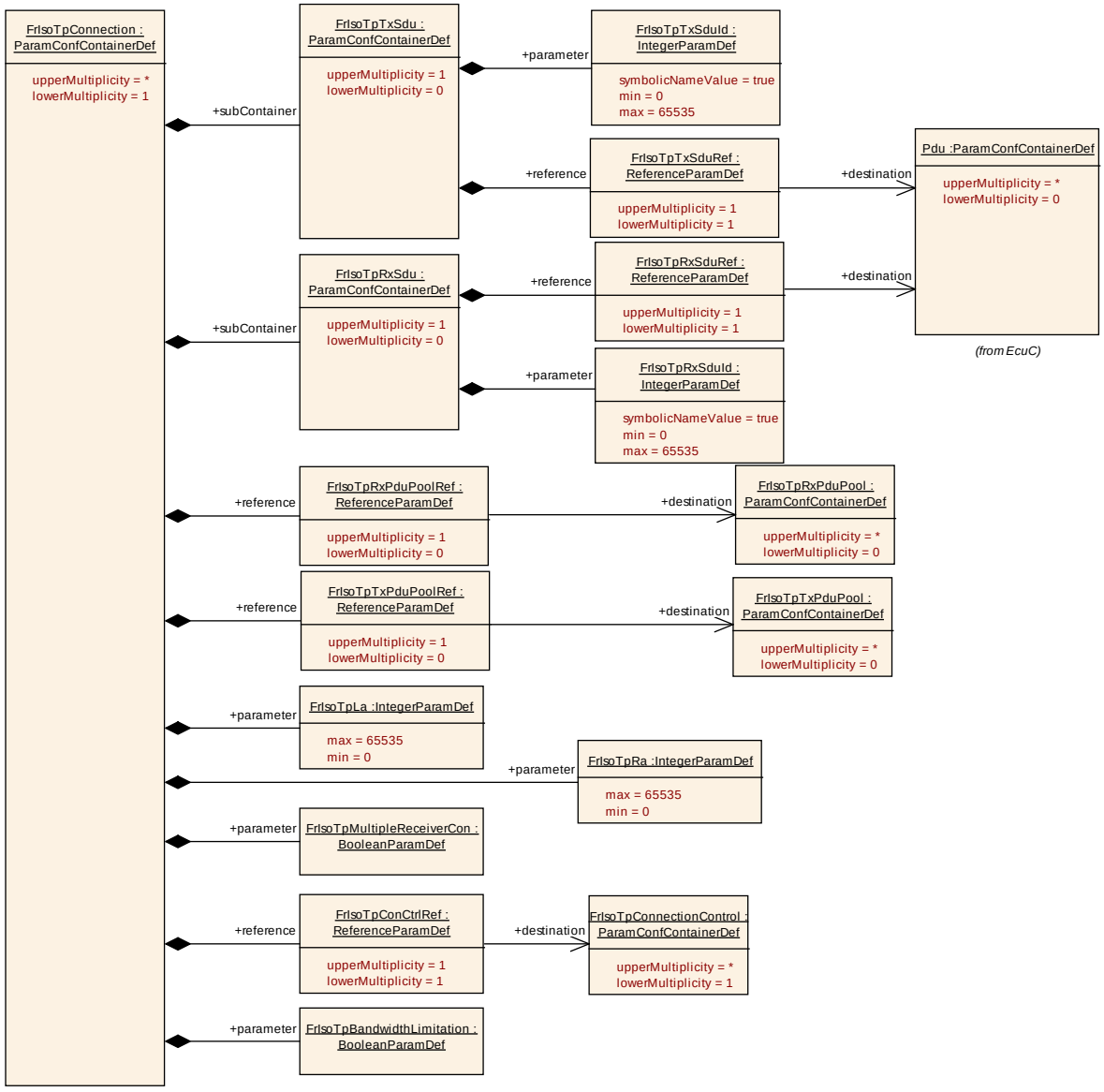
	When the local instance is the receiver, this is the Source Address within the TP frame.		
Multiplicity	1		
Type	IntegerParamDef		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**FRISOTP005_Conf :**		
Name	FrIsoTpConCtrlRef {FRISOTP_CON_CTRL_REF}		
Description	FrIsoTpConnectionControlReference: This parameter defines a reference to a connection control container.		
Multiplicity	1		
Type	Reference to [FrIsoTpConnectionControl]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**FRISOTP025_Conf :**		
Name	FrIsoTpRxPduPoolRef {FRISOTP_RX_PDU_POOL_REF}		
Description	This parameter defines a reference to a RxPduPool.		
Multiplicity	0..1		
Type	Reference to [FrIsoTpRxPduPool]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**FRISOTP039_Conf :**		
Name	FrIsoTpTxPduPoolRef {FRISOTP_TX_PDU_POOL_REF}		
Description	This parameter defines a reference to a TxPduPool.		
Multiplicity	0..1		
Type	Reference to [FrIsoTpTxPduPool]		
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***
FrIsoTpRxSdu	0..1	This parameter defines the Rx Service Data Unit Identifier (Sdu Id) which uniquely identifies a data transfer (inter-module communication) between FrIsoTp and PDUR.
FrIsoTpTxSdu	0..1	This parameter defines the Tx Service Data Unit Identifier (Sdu Id) which uniquely identifies a data transfer (inter-module communication) between FrIsoTp and PDUR.



10.2.6 FrIsoTpTxSdu

SWS Item	**FRISOTP041_Conf :**
Container Name	FrIsoTpTxSdu
Description	This parameter defines the Tx Service Data Unit Identifier (Sdu Id) which uniquely identifies a data transfer (inter-module communication) between FrIsoTp and PDUR.
Configuration Parameters	

SWS Item	**FRISOTP042_Conf :**		
Name	FrIsoTpTxSduId {FRISOTP_SDUID}		
Description	This is a unique identifier for a to be transmitted message from the PduR to the FrIsoTp. ImplementationType: PduIdType		
Multiplicity	1		
Type	IntegerParamDef (Symbolic Name generated for this parameter)		
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	**FRISOTP043_Conf :**		
Name	FrIsoTpTxSduRef		
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

10.2.7 FrIsoTpRxSdu

SWS Item	**FRISOTP027_Conf :**
Container Name	FrIsoTpRxSdu
Description	This parameter defines the Rx Service Data Unit Identifier (Sdu Id) which uniquely identifies a data transfer (inter-module communication) between FrIsoTp and PDUR.
Configuration Parameters	

SWS Item	**FRISOTP052_Conf :**		
Name	FrIsoTpRxSduId {FRISOTP_SDUID}		
Description	This is a unique identifier for a received message. This Id is used in the CancelReceive API call. ImplementationType: PduIdType		
Multiplicity	1		
Type	IntegerParamDef (Symbolic Name generated for this parameter)		
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	**FRISOTP028_Conf :**		
Name	FrIsoTpRxSduRef		
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

Note: *FrIsoTpRxSduId* is actually not used by CancelReceive API.

10.2.8 FrIsoTpConnectionControl

SWS Item	**FRISOTP007_Conf :**
Container Name	FrIsoTpConnectionControl{FrIsoTpConnectionControl}
Description	This container contains the configuration parameters to control a FlexRay ISO TP connection.
Configuration Parameters	

SWS Item	**FRISOTP003_Conf :**		
Name	FrIsoTpAckType {FRISOTP_ACKTYPE}		
Description	This parameter defines the type of acknowledgement which is used for the specific channel.		
Multiplicity	1		
Type	EnumerationParamDef		
Range	FRISOTP_ACK_WITH_RT	Acknowledgement with retry	
	FRISOTP_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	**FRISOTP012_Conf :**		
Name	FrIsoTpMaxAr {FRISOTP_MAX_AR}		
Description	This parameter defines the maximum number of trying to send a frame when a TIMEOUT AR occurs.		
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	**FRISOTP013_Conf :**		
Name	FrIsoTpMaxAs {FRISOTP_MAX_AS}		
Description	This parameter defines the maximum number of trying to send a frame when a TIMEOUT AS occurs.		
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	**FRISOTP014_Conf :**		
Name	FrIsoTpMaxFCWait {FRISOTP_MAX_FCWAIT}		
Description	This parameter defines the maximum number of FlowControl N-PDUs with FlowState "WAIT"		
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	**FRISOTP016_Conf :**		
Name	FrIsoTpMaxFrIf {FRISOTP_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	**FRISOTP029_Conf :**		
Name	FrIsoTpMaxNbrOfNPduPerCycle {FRISOTP_MAX_NUMBER_OF_NPDU_PER_CYCLE}		
Description	This parameter is part of the ISO 10681-2 protocol's FlowControl parameter "Bandwidth Control (BC)". It limits the number of N-Pdus the sender is allowed to transmit within a FlexRay cycle.		
Multiplicity	1		
Type	IntegerParamDef		
Range	0 .. 31		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**FRISOTP017_Conf :**		
Name	FrIsoTpMaxRn {FRISOTP_MAX_RN}		
Description	This parameter defines the maximum number of retries (if 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	**FRISOTP020_Conf :**		
Name	FrIsoTpSCexp {FRISOTP_SEPARATION_CYCLE_EXPONENT}		
Description	This parameter is part of the ISO 10681-2 protocol's FlowControl parameter "Bandwidth Control (BC)". It represents the exponent to calculate the minimum number of "Separation Cycles" the sender has to wait for the next transmission of an FrIsoTp N-Pdu.		
Multiplicity	1		
Type	IntegerParamDef		
Range	0 .. 7		
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	**FRISOTP047_Conf :**		
Name	FrIsoTpTimeBr {FRISOTP_TIME_BR}		
Description	This parameter defines the time in seconds the FrIsoTp requires to transmit a corresponding FlowControl Frame. According to ISO 10681-2 this parameter is a performance requirement.		
Multiplicity	1		
Type	FloatParamDef		
Range	0 .. 0.255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	**FRISOTP031_Conf :**		
Name	FrIsoTpTimeFrIf {FRISOTP_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 .. 0.255		
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	**FRISOTP032_Conf :**		
Name	FrIsoTpTimeoutAr {FRISOTP_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 .. 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		

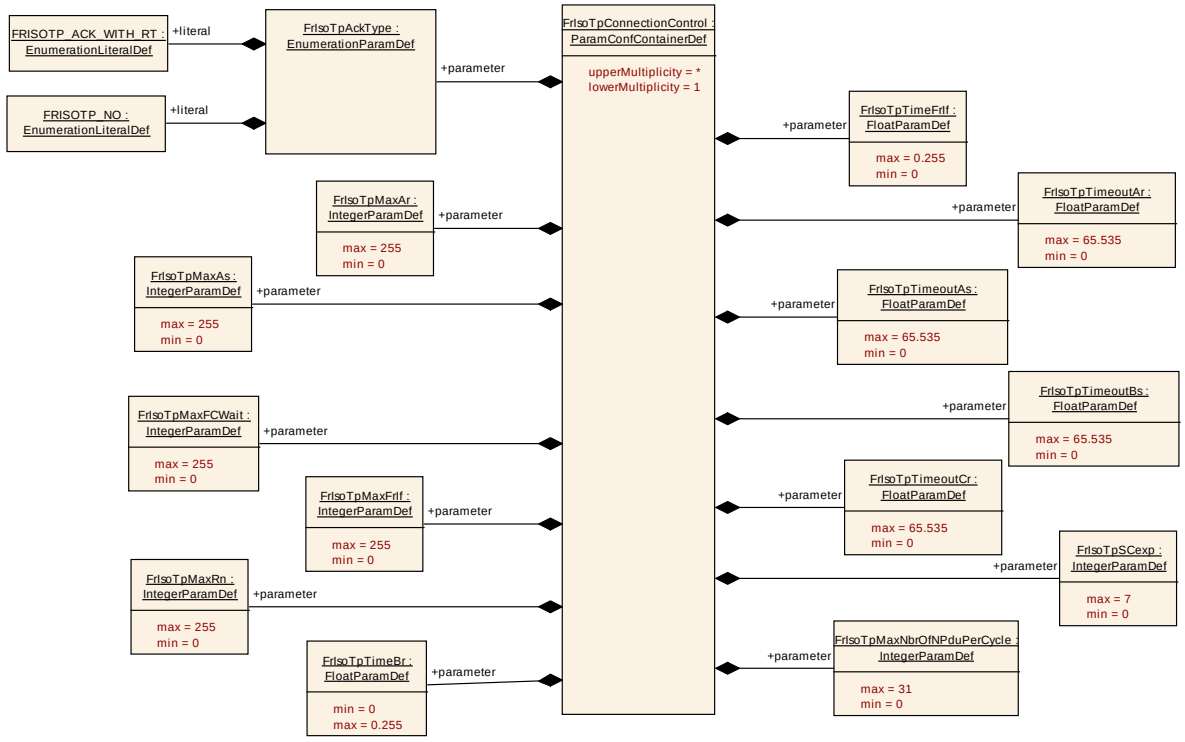
SWS Item	**FRISOTP033_Conf :**		
Name	FrIsoTpTimeoutAs {FRISOTP_TIMEOUT_AS}		
Description	This parameter specifies the timeout in seconds the FrIf shall confirm a transmitted Pdu to the FrIsoTp.		
Multiplicity	1		
Type	FloatParamDef		
Range	0 .. 65.535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

	dependency: It is obvious that FRISOTP_TIME_CS + FRISOTP_TIMEOUT_AS < FRISOTP_TIMEOUT_CR must hold (because the transmission duration on the bus has also to be considered).

SWS Item	**FRISOTP034_Conf :**		
Name	FrIsoTpTimeoutBs {FRISOTP_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 .. 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 dependency: It is obvious that FRISOTP_TIME_BR + FRISOTP_TIMEOUT_AR < FRISOTP_TIMEOUT_BS must hold (because the transmission duration on the bus has also to be considered).		

SWS Item	**FRISOTP035_Conf :**		
Name	FrIsoTpTimeoutCr {FRISOTP_TIMEOUT_CR}		
Description	This parameter defines the timeout value in seconds a receiver is waiting for a CF or a LF.		
Multiplicity	1		
Type	FloatParamDef		
Range	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 dependency: It is obvious that FRISOTP_TIME_CS + FRISOTP_TIMEOUT_AS < FRISOTP_TIMEOUT_CR must hold (because the transmission duration on the bus has also to be considered).		

No Included Containers



10.2.9 FrIsoTpTxPduPool

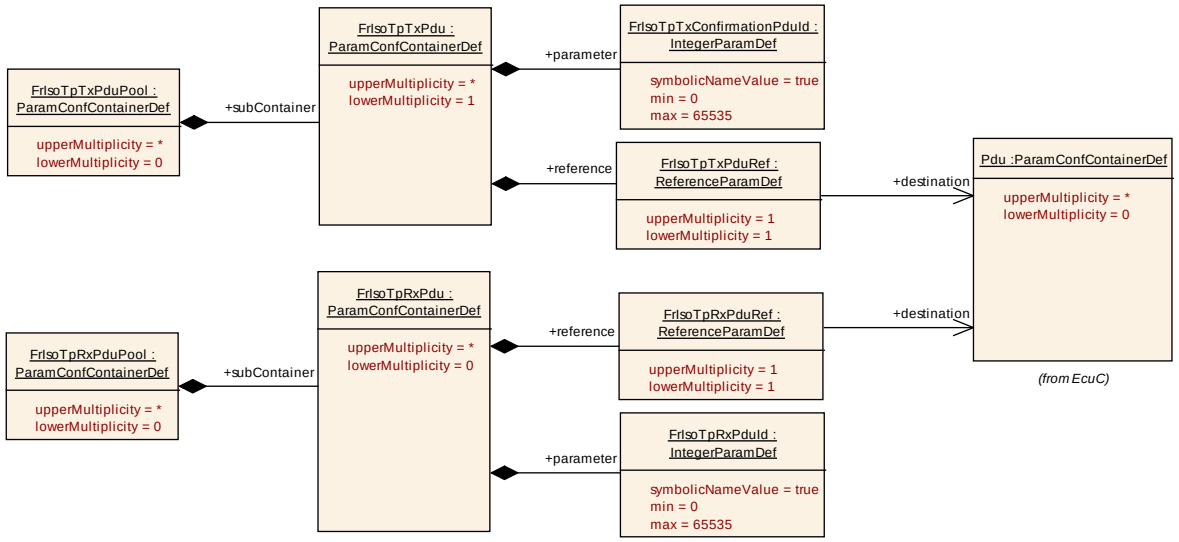
SWS Item	**FRISOTP038_Conf :**
Container Name	FrIsoTpTxPduPool{PduPoolContainer}
Description	This container contains all Pdus that are assigned to that Pdu Pool.
Configuration Parameters	

Included Containers		
Container Name	*Multiplicity*	*Scope / Dependency*
FrIsoTpTxPdu	1..*	Container to hold the PDU parameters. ImplementationType: PduInfoType

10.2.10 FrIsoTpRxPduPool

SWS Item	**FRISOTP024_Conf :**
Container Name	FrIsoTpRxPduPool{PduPoolContainer}
Description	This container contains all Pdus that are assigned to that Pdu Pool.
Configuration Parameters	

Included Containers		
Container Name	*Multiplicity*	*Scope / Dependency*
FrIsoTpRxPdu	0..*	Container to hold the PDU parameters. ImplementationType: PduInfoType



10.2.11 FrIsoTpTxPdu

SWS Item	**FRISOTP037_Conf :**
Container Name	FrIsoTpTxPdu
Description	Container to hold the PDU parameters. ImplementationType: PduInfoType
Configuration Parameters	

SWS Item	**FRISOTP049_Conf :**		
Name	FrIsoTpTxConfirmationPduId		
Description	Handle Id to be used by the FrIf to confirm the transmission of the FrIsoTpTxPdu to the FrIf module.		
Multiplicity	1		
Type	IntegerParamDef (Symbolic Name generated for this parameter)		
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	**FRISOTP040_Conf :**		
Name	FrIsoTpTxPduRef		
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

10.2.12 FrIsoTpRxPdu

SWS Item	**FRISOTP022_Conf :**

Container Name	FrIsoTpRxPdu
Description	Container to hold the PDU parameters. ImplementationType: PduInfoType
Configuration Parameters	

SWS Item	**FRISOTP023_Conf :**		
Name	FrIsoTpRxPduId		
Description	This is a unique identifier for a received message which is forwarded from the FrIf to the FrIsoTp. ImplementationType: PduIdType		
Multiplicity	1		
Type	IntegerParamDef (Symbolic Name generated for this parameter)		
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	**FRISOTP026_Conf :**		
Name	FrIsoTpRxPduRef		
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

10.3 Published Information

FRISOTP001_PI: The standardized common published parameters as required by BSW00402 in the General Requirements on Basic Software Modules [3] shall be published within the header file of this module and need to be provided in the BSW Module Description. The according module abbreviation can be found in the List of Basic Software Modules [1].

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

10.4 Configuration dependencies and recommendation

The FrIsoTp module functionality is based on several configuration parameters. To guarantee a well working software module this chapter gives some recommendation to the configuration parameter set. This rules shall be part of consistency checks of configuration tools.

10.4.1 Retry behaviour

The term of retry is used several times within this document but always with different focus. As depict in Figure 25 the FrIsoTp module has basically two different retry behaviours:

a) Multiple API calls in case of an error or a busy system
b) Retry of PDU transfer depending on transport protocol conditions.

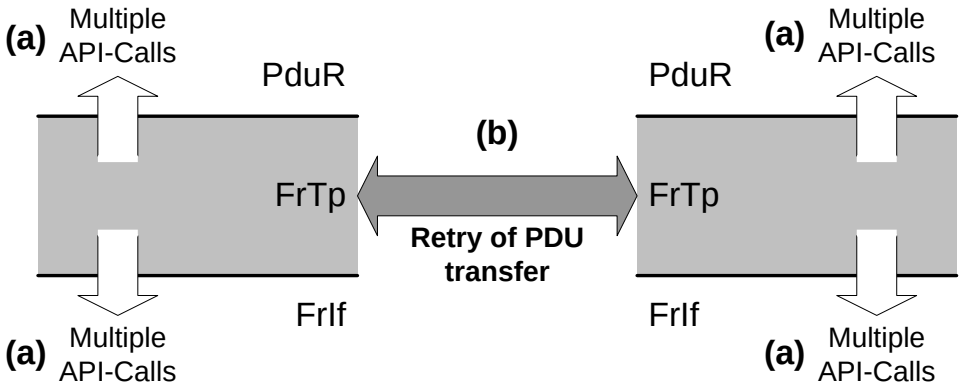


Figure 25: FrIsoTp Retry Scenarios

Only for case (b) Retry of PDU transfer a global switch for enabling and disabling is defined (FRISOTP_ ACKTYPE). All other cases depend on the configuration of the different counters for the API calls:
FRISOTP_MAX_AR

10.4.2 TP-Acknowledgement and Retry

Acknowledgement and retry is only possible on 1:1 connections because the communication nodes have to deal with the FlowControl N-PDU parameters. FlowControl is not allowed for 1:n connections.

FRISOTP598: If configuration parameter *FrIsoTpMultipleReceiverCon* is set for a connection, no Acknowledgement and retry shall be supported irrespective whether the configuration parameter *FRISOTP_ACKTYPE* is set or not.

10.4.3 Timing and Timeout Parameters

Timing and timeout behaviour depends on the global FlexRay schedule.
To guarantee a stable system some timing and timeout relations shall be taken into account. The timeout behaviour is defined in chapter 7.5.8.

→ *Configuration Notes*

FRISOTP1154: For timeout As configuration it shall be considered that
FRISOTP_TIMEOUT_AS > FRISOTP_TIME_AS

FRISOTP1155: For timeout Ar configuration it shall be considered that
FRISOTP_TIMEOUT_AR > FRISOTP_TIME_AR

FRISOTP599: For timeout Bs configuration it shall be considered that
FRISOTP_TIMEOUT_BS > FRISOTP_TIME_BR + FRISOTP_TIME_AR

FRISOTP1153: For timeout Cr configuration it shall be considered that
FRISOTP_TIMEOUT_CR > FRISOTP_TIME_CS + FRISOTP_TIME_AS

Note: *FRISOTP_TIME_AR*, *FRISOTP_TIME_AS*, *FRISOTP_TIME_BR* and *FRISOTP_TIME_CS* are performance timing values and depend on the global FlexRay schedule. To calculate that values please refer to the formulas at ISO 10681-2 [16]

FRISOTP180: The FrIsoTp configuration shall ensure that $2^{SeparationCycleExponent}-1 \times t_{CycleTime} \leq$ FRISOTP_TIMEOUT_CR .[36]

[36] This is to prevent a timeouts. Please refer to ISO 10681-2.

10.4.4 Configuration Requirements on the FlexRay ISO Transport Protocol Layer

FRISOTP1171: The parameters `FRISOTP_MAX_RN` shall have the same value in the sender and the receiver peer[37].

10.4.5 Bandwidth Control Configuration

It could occur that an ECU is not able to receive as much PDUs as defined within the PDU-Pool referenced by a connection[38]. In that case the bandwidth has to be limited by the receiver. There are three possibilities to limit the bandwidth for a connection link:

a) Limit Bandwidth by Parameter FlowControl.BandwidthControl.MaxPduPerCycle in combination with Hardware FIFO buffer mechanisms.[39]
b) Limit Bandwidth by Parameter FlowControl.BandwidthControl.MaxPduPerCycle to reduce the number of allowed PDUs of the currently selected PDU-Pool.
c) Limit Bandwidth by using a dedicated PDU-Pool for the connection to the affected Ecu.

[37] Refer to 44
[38] This is possible if a Flexray Communication Controller only supports less Rx buffers and buffer reconfiguration at the end of a cycle is not possible or desired.
[39] This is an esential mechanism of FlexRay 3.0 protocol.

10.4.5.1 BandwidthControl by FlowControl Parameter in combination with Hardware FIFO buffer mechanisms

If BandwidthControl by FlowControl Parameter in combination with Hardware FIFO buffer mechanisms is used no additional configuration restictions occur. This szenario implements “pure” ISO 10681-2 behaviour with focus on FlexRay 3.0 Hardware FIFO buffer mechanisms. The figure below shows the dependencies.

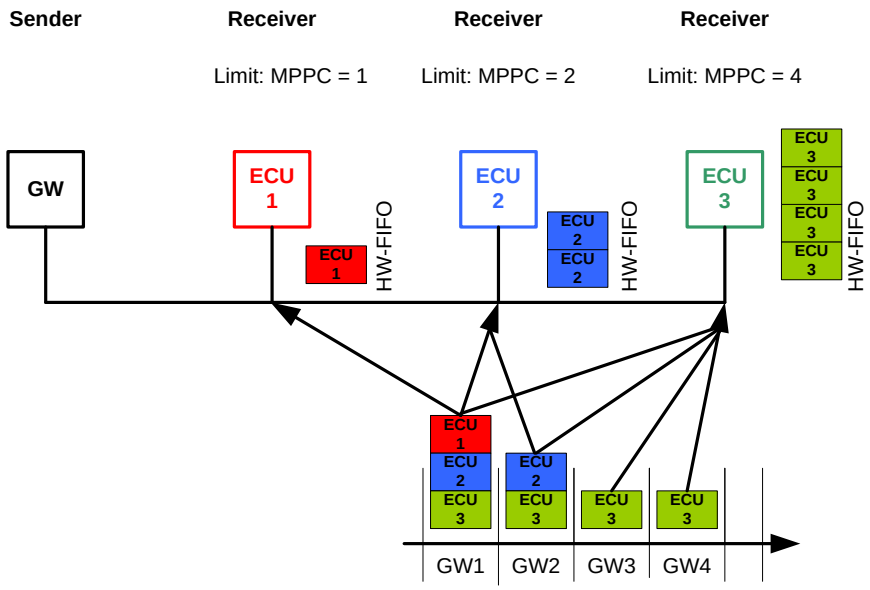
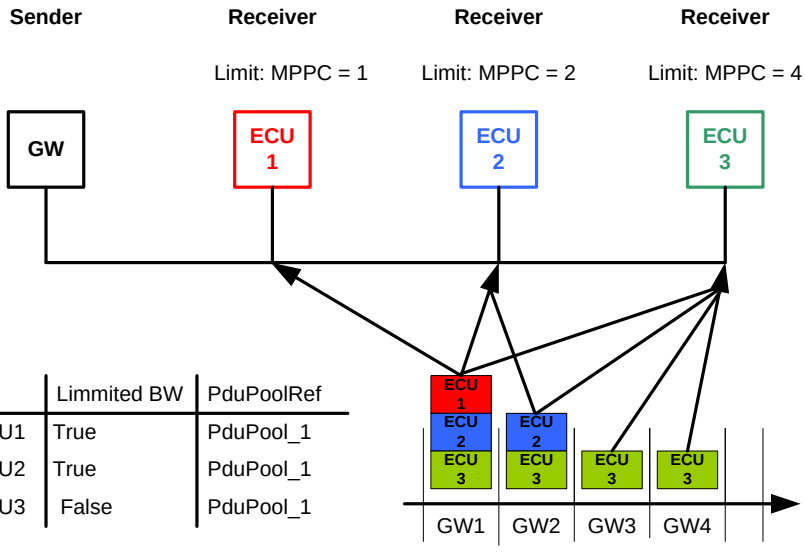


Figure 26: BandwidthControl by FlowControl Parameters in combination with HW FIFO buffer

10.4.5.2 BandwidthControl by FlowControl Parameter

If BandwidthControl by FlowControl Parameter is used some configuration restictions have to be taken into account. The figure below shows the dependencies.



Connection	Limmited BW	PduPoolRef
Connection_ECU1	True	PduPool_1
Connection_ECU2	True	PduPool_1
Connection_ECU3	False	PduPool_1

Figure 27: BandwidthControl by FlowControl Parameters

In a network four FlexRay nodes are connected. 4 PDUs are defined for the sender node. Two of the receivers have bandwidth limitations (ECU1 and ECU2).
The configuration restrictions are:

FRISOTP1173: If bandwidth limitation in a connection to a certain ECU is realized by FlowControl parameter BC (without HW-FIFO mechanism), the attribute “FrIsoTpBandwidthLimitation“within the corresponding connection shall be set to „TRUE“.

FRISOTP1174: If bandwidth limitation is realized by FlowControl parameter BC and if the attribute “FrIsoTpBandwidthLimitation“ is True, a Start Frame to initiate a communication link shall always be send in the first PDU of the referenced PDU-Pool. This is valid for both 1:1 and 1:n connections.

FRISOTP1175: If an ECU responds with a FlowControl-Parameter BandwidthControl. MPPC ≠ 0 (“zero”) the sender shall use only the number of BC.MPPC PDUs of the PDU-Pool in ascending order to transmit data within that connection.

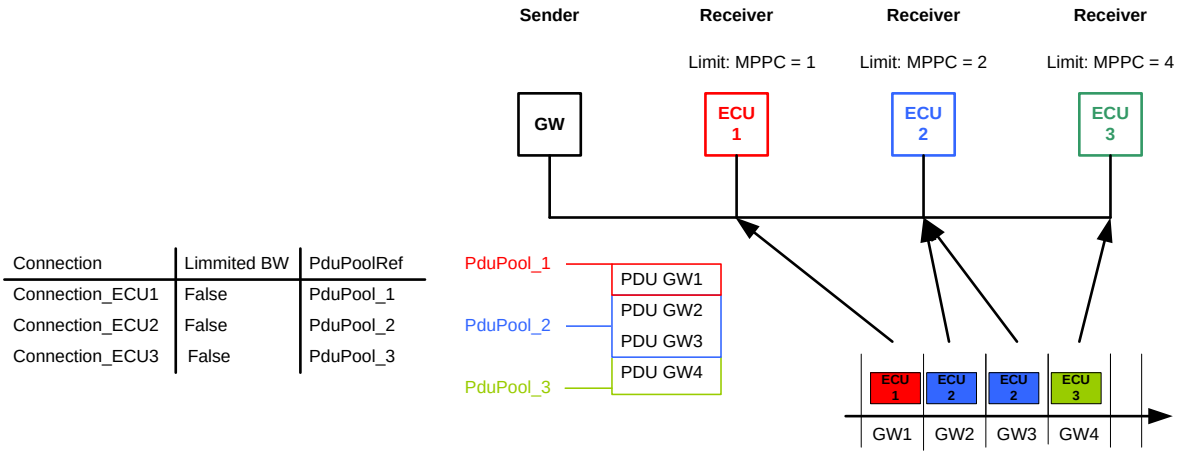
FRISOTP1177: If the attribute “FrIsoTpBandwidthLimitation“ is set to "TRUE", a Rx-connection shall use the first Pdu of the referenced Tx-Pdu-Pool for sending the required FlowControl frame to continue a communication link.

10.4.5.3 BandwidthControl via different PDU Pools

If BandwidthControl is realized by different PDU Pools two different szenarios could occur.

10.4.5.3.1 BandwidthControl via non-overlapping Tx-Pdu-Pools

In case an ECU is not capable of receiving all Tx-Pdus sent for TP-communication in the FlexRay-cluster then non-overlapping Tx-Pdu-Pools can be configured as shown in the figure below:

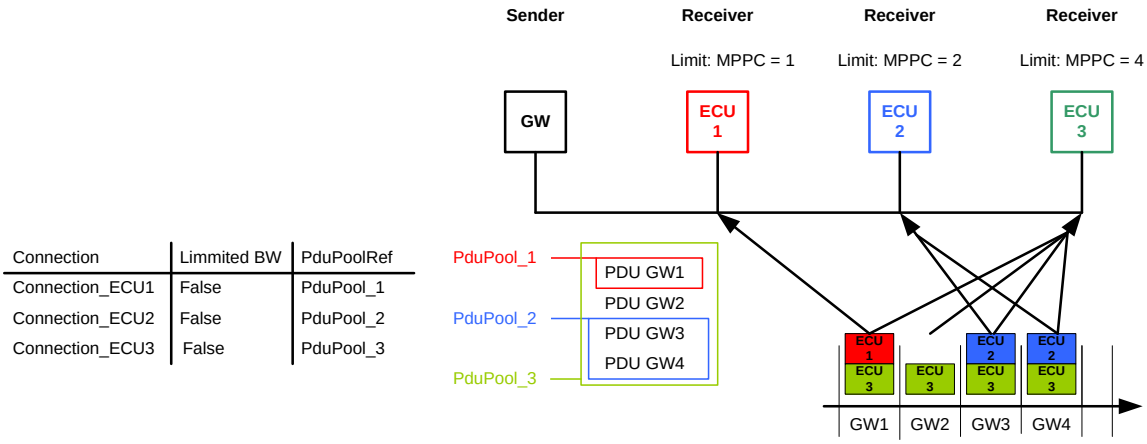


Connection	Limmited BW	PduPoolRef
Connection_ECU1	False	PduPool_1
Connection_ECU2	False	PduPool_2
Connection_ECU3	False	PduPool_3

Figure 28: BandwidthControl by non-overlapping PDU Pools

10.4.5.3.2 BandwidthControl via overlapping Tx-Pdu-Pools

In case an Ecu is not capable of receiving all Tx-Pdus sent for TP- communication in the FlexRay-cluster then dedicated overlapping Tx-Pdu-Pools can be configured as shown in the figure below:



Connection	Limmited BW	PduPoolRef
Connection_ECU1	False	PduPool_1
Connection_ECU2	False	PduPool_2
Connection_ECU3	False	PduPool_3

Figure 29: BandwidthControl by multiple PDU Pools

In the figure above ECU1 and ECU2 are not capable of receiving all Pdus GW1-GW4 shown above in their Rx-buffers ("Weak Ecu") and dedicated overlapping Pdu-Pools are configured and used. One Tx-Pdu can belong to more than one Tx-PDU-Pool at the same time[40].

FRISOTP1176: It shall be possible to have overlapping PDU-Pools

10.4.6 Configuration Requirements on the FlexRay Interface

[40] Reduced pools have to be taken into account for configuring the FlexRay-driver of "weak Ecus".

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 ISO Transport Protocol Layer uses them, i.e. in ascending order regarding the Fr N-PDU IDs used in the FlexRay ISO Transport Protocol Layer. Furthermore these PDUs shall be scheduled with the same frequency and within one Job (concerning the Joblist) in the FlexRay Interface module (since the reading of the PDU-Available Information for all PDUs of a connection has to be atomic.) This is necessary to avoid CFs coming out of order in a segmented transfer.

For every FrIsoTp L-SDU the PDU-Update/Valid Information of the FrIf shall be activated.

For each transmitted N-Pdu a TransmitConfirmations shall be given by the FrIf module.

For every FrIsoTp 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 service primitive 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.