Document Title	Specification of CAN Transport Layer
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	14

Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R21-11

Document Change History			
Date	**Release**	**Changed by**	**Description**
2021-11-25	R21-11	AUTOSAR Release Management	• Improve Error handling • Clarifications
2020-11-30	R20-11	AUTOSAR Release Management	• Improve Error sections • Clarifications
2019-11-28	R19-11	AUTOSAR Release Management	• Added configuration diagrams • Clarifications • Changed Document Status from Final to published
2018-10-31	4.4.0	AUTOSAR Release Management	• Removed some limitations for Half-duplex • Minor corrections
2017-12-08	4.3.1	AUTOSAR Release Management	• Clarification of metadata provision • Extend data length for CAN-FD • Rollout of Runtime errors • Minor corrections
2016-11-30	4.3.0	AUTOSAR Release Management	• Harmonized API functions description • Parallel handling of CAN 2.0 and CAN-FD clarification • Introduction of reliable Tx Confirmation • Clarification of addressing in Upper Layers using MetaData

2015-07-31	4.2.2	AUTOSAR Release Management	• File structure correction • FC_OVFL clarification • DET Renaming and Extension Incorporation
2014-10-31	4.2.1	AUTOSAR Release Management	• Introduced support for CAN Flexible Data rate • Minor corrections • Clarifications
2014-03-31	4.1.3	AUTOSAR Release Management	• Revised padding behaviour. • Clarified relation between CanTp MainFunctionPeriod and other timers. • Revised CanTp_RxIndication() prototype. • Extended parameter CanTpTc for receive cancellation.
2013-10-31	4.1.2	AUTOSAR Release Management	• Replace NTFRSLT_OK\NTFRSLT_<other> E_OK\E_NOT_OK • Handling of unexpected arrival of N-PDU table clarification • Editorial changes • Removed chapter(s) on change documentation
2013-03-15	4.1.1	AUTOSAR Release Management	• Error handling has been improved • PostBuild concept has been refined • Introduction of HDV support • Clarifications of buffer handling
2011-12-22	4.0.3	AUTOSAR Release Management	• CanTp does not report production errors anymore • Metamodel structure changed • Harmonization with the new buffer concept • Change the BlockSize to be statically configurable instead a maximum value
2011-04-15	4.0.2	AUTOSAR Release Management	• Corrections and improvement in errors description; • API services correction; • Clarifications in relation with buffer handling • Updated table in Ch. 6 for half and full duplex support

2009-12-18	4.0.1	AUTOSAR Release Management	• Added Mixed Addressing Mode • CanTp supports Full Duplex Mode • New buffering concept • Added possibility to change CanTp parameters • Legal disclaimer revised
2007-12-21	3.0.1	AUTOSAR Release Management	• Addition of transmit cancellation feature • DataLength check only for too small DLC (CanTp220) • Restriction on mapping of N-Pdu (SWS_CanTp_00248) • Document meta information extended • Small layout adaptations made
2007-07-24	2.1.16	AUTOSAR Release Management	• “Advice for users” revised • “Revision Information” added
2007-01-24	2.1.15	AUTOSAR Release Management	• Clarification and correction of error management: list of production\development error and behavior in case of error • Addition of SWS_CanTp_00166 and SWS_CanTp_00167 to avoid blocking situation in case of no buffer provided by upper layer • Remove of CanTpRxWftMax of container CanTpTxNSdu • 1 parameter added for the call of Det_ReportError • Add header files inclusions • Addition of CanTpNSa container in configuration chapter • Legal disclaimer revised
2006-11-28	2.1	AUTOSAR Release Management	• Document structure adapted to common Release 2.0 SWS Template.
2005-05-31	1.0	AUTOSAR Release Management	• Initial Release

Disclaimer

Table of Contents

1 Introduction and functional overview

This specification defines the functionality, API and the configuration of the AUTOSAR Basic Software module CAN Transport Layer (CanTp).

CanTp is the module between the PDU Router and the CAN Interface module (see Figure 1.1). The main purpose of the CAN TP module is to segment and reassemble CAN I-PDUs longer than 8 bytes or longer than 64 bytes in case of CAN FD.

The PDU Router deploys AUTOSAR COM and DCM I-PDUs onto different communication protocols. The routing through a network system type (e.g. CAN, LIN and Flex Ray) depends on the I-PDU identifier. The PDU Router also determines if a transport protocol has to be used or not. Lastly, this module carries out gateway functionality, when there is no rate conversion.

CAN Interface (CanIf) provides equal mechanisms to access a CAN bus channel regardless of its location (μC internal/external). From the location of CAN controllers (on chip / onboard), it extracts the ECU hardware layout and the number of CAN drivers. Because CanTp only handles transport protocol frames (i.e. SF, FF, CF and FC PDUs), depending on the N-PDU ID, the CAN Interface has to forward an I-PDU to CanTp or PduR.

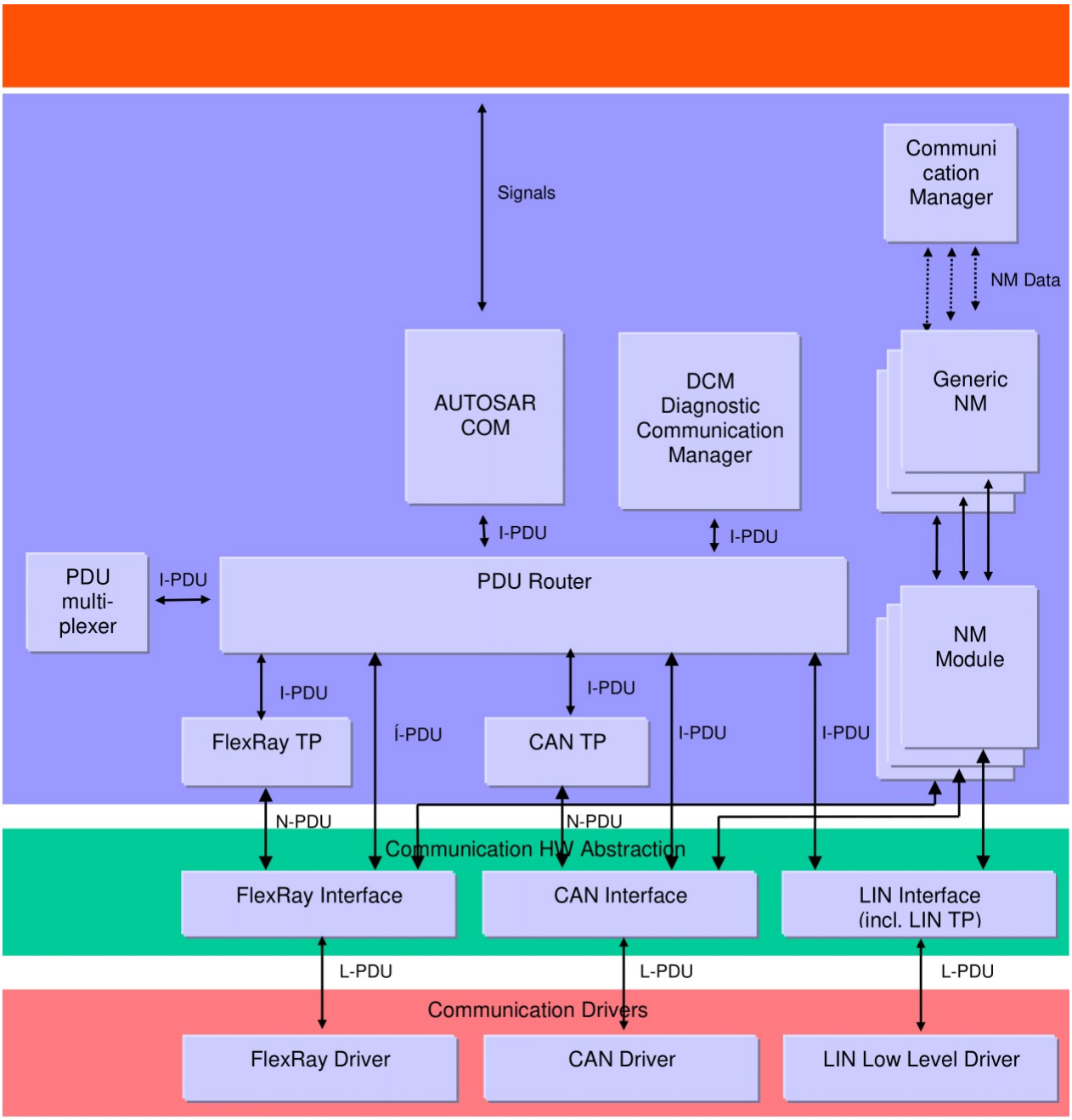

Figure 1.1: AUTOSAR Communication Stack

According to AUTOSAR basic software architecture, CanTp provides services for:

- Segmentation of data in transmit direction
- Reassembling of data in receive direction
- Control of data flow
- Detection of errors in segmentation sessions
- Transmit cancellation
- Receive cancellation

It is an AUTOSAR decision to base basic software module specifications on existing standards, thus this AUTOSAR CAN Transport Layer specification is based on the international standard ISO 15765, which is the most used standard in the automotive domain.

ISO 15765 (containing four sections) describes two applicable CAN Transport Layer specifications: ISO 15765-2 for OEM enhanced diagnostics [1] and ISO 15765-4 for OBD diagnostics [2]. Concerning the transport layer, ISO 15765-4 (the section of ISO 15765 which also covers the data link layer and physical layer) is in accordance with ISO 15765-2 with some restrictions/additions. In order that there is no incompatibility problem between ISO 15765-2 and ISO 15765-4, differences will be solved by the CAN Transport Layer configuration.

Although CAN transport protocol is mainly used for vehicle diagnostic systems, it has also been developed to deal with requirements from other CAN based systems requiring a transport layer protocol.

2 Acronyms and Abbreviations

The glossary below includes acronyms and abbreviations relevant to the CAN Transport Layer module that are not included in the [3, AUTOSAR glossary].

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

Prefix:	*Description:*
I-	Relative to AUTOSAR COM **I**nteraction Layer
L-	Relative to the CAN Interface module which is equivalent to the **L**ogical Link Control (the upper part of the Data Link Layer - the lower part is called Media Access Control)
N-	Relative to the CAN Transport Layer which is equivalent to the OSI **N**etwork Layer.

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

Acronym:	*Description:*
CAN L-SDU	This is the SDU of the CAN Interface module. It is similar to CAN N-PDU but from the CAN Interface module point of view.
CAN LSduId	This is the unique identifier of a SDU within the CAN Interface. It is used for referencing L-SDU's routing properties. Consequently, in order to interact with the CAN Interface through its API, an upper layer uses CAN LSduId to refer to a CAN L-SDU Info Structure.
CAN N-PDU	This is the PDU of the CAN Transport Layer. It contains a unique identifier, data length and data (protocol control information plus the whole N-SDU or a part of it).
CAN N-SDU	This is the SDU of the CAN Transport Layer. In the AUTOSAR architecture, it is a set of data coming from the PDU Router.
CAN N-SDU Info Structure	This is a CAN Transport Layer internal constant structure that contains specific CAN Transport Layer information to process transmission, reception, segmentation and reassembly of the related CAN N-SDU.

△

Acronym:	*Description:*
CAN NSduId	Unique SDU identifier within the CAN Transport Layer. It is used to reference N-SDU's routing properties. Consequently, to interact with the CAN Transport Layer via its API, an upper layer uses CAN NSduId to refer to a CAN N-SDU Info Structure.
I-PDU	This is the PDU of the AUTOSAR COM module.
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).
SDU	In layered systems, this refers to a set of data that is sent by a user of the services of a given layer, and is transmitted to a peer service user, whilst remaining semantically unchanged.

Abbreviation:	*Description:*
BS	Block Size
Can	CAN Driver module
CAN CF	CAN Consecutive Frame N-PDU
CAN FC	CAN Flow Control N-PDU
CAN FF	CAN First Frame N-PDU
CAN SF	CAN Single Frame N-PDU
CanIf	CAN Interface
CanTp	CAN Transport Layer
CanTrcv	CAN Transceiver module
CF	See "CAN CF"
Com	AUTOSAR COM module
Dcm	Diagnostic Communication Manager module
DEM	Diagnostic Event Manager
DET	Default Error Tracer
DLC	Data Length Code (part of CAN PDU that describes the SDU length)
FC	See "CAN FC"
FF	See "CAN FF"
FIM	Function Inhibition Manager
Mtype	Message Type (possible value: diagnostics, remote diagnostics)
N_AI	Network Address Information (see ISO 15765-2).
N_Ar	Time for transmission of the CAN frame (any N-PDU) on the receiver side (see ISO 15765-2 [1]).
N_As	Time for transmission of the CAN frame (any N-PDU) on the sender side (see ISO 15765-2 [1]).
N_Br	Time until transmission of the next flow control N-PDU (see ISO 15765-2 [1]).
N_Bs	Time until reception of the next flow control N-PDU (see ISO 15765-2 [1]).
N_Cr	Time until reception of the next consecutive frame N-PDU (see ISO 15765-2 [1]).
N_Cs	Time until transmission of the next consecutive frame N-PDU (see ISO 15765-2 [1]).
N_Data	Data information of the transport layer
N_PCI	Protocol Control Information of the transport layer
N_SA	Network Source Address (see ISO 15765-2 [1]).
N_TA	Network Target Address (see ISO 15765-2 [1]). It might already contain the N_TAtype(physical/function) in case of ExtendedAddressing.
N_TAtype	Network Target Address type (see ISO 15765-2 [1]).

▽

△

Abbreviation:	*Description:*
OBD	On-Board Diagnostic
PDU	Protocol Data Unit
PduR	PDU Router
SDU	Service Data Unit
FS	Flow Status
CAN FD	CAN flexible data rate
CAN_DL	CAN frame data length
TX_DL	Transmit data link layer data length
RX_DL	Received data link layer data length
SF_DL	SingleFrame data length in bytes

The following table contains some of the concepts, which are useful in this work:

Definitions:	*Description:*
Default Error Tracer	The Default Error Tracer is merely a support to SW development and integration and is not contained in the production code. The API is defined, but the functionality can be chosen and implemented by the developer according to his specific needs.
Diagnostic Event Manager	The Diagnostic Event Manager is a standard AUTOSAR module which is available in the production code and whose functionality is specified in the AUTOSAR project.
Extended addressing format	A unique CAN identifier is assigned to each combination of N_SA and Mtype. A unique address is filed to each combination of N_TA and N_TAtype in the first data byte of the CAN frame data field. N_PCI and N_Data are filed in the remaining bytes of the CAN frame data field.
Function Inhibition Manager	The Function Inhibition Manager (FIM) stands for the evaluation and assignment of events to the required actions for Software Components (e.g. inhibition of specific “monitoring functions”). The DEM informs and updates the Function Inhibition Manager (FIM) upon changes of the event status in order to stop or release functional entities according to assigned dependencies. An interface to the functional entities is defined and supported by the Mode Manager. The FIM is not part of the DEM.
Functional addressing	In the transport layer, functional addressing refers to N-SDU, of which parameter N_TAtype (which is an extension to the N_TA parameter [1] used to encode the communication model) has the value functional. This means the N-SDU is used in 1 to n communications. Thus with the CAN protocol, functional addressing will only be supported for Single Frame communication. In terms of application, functional addressing is used by the external (or internal) tester if it does not know the physical address of an ECU that should respond to a service request or if the functionality of the ECU is implemented as a distributed server over several ECUs. When functional addressing is used, the communication is a communication broadcast from the external tester to one or more ECUs (1 to n communication). Use cases are (for example) broadcasting messages, such as “ECUReset” or “Communication Control” OBD communication will always be performed as part of functional addressing.
Mixed addressing format	A unique CAN identifier is assigned to each combination of N_SA, N_TA, N_TAtype. N_AE is placed in the first data byte of the CAN frame data field. N_PCI and N_Data are placed in the remaining bytes of the CAN frame data field.
Multiple connection	The CAN Transport Layer should manage several transport protocol communication sessions at a time.
Normal addressing format	A unique CAN identifier is assigned to each combination of N_SA, N_TA, N_TAtype and Mtype. N_PCI and N_Data are filed in the CAN frame data field.

▽

△

Definitions:	*Description:*
Physical addressing	In the transport layer, physical addressing refers to N-SDU, of which parameter N_TAtype (which is an extension of the N_TA parameter [1] used to encode the communication model) has the value physical. This means the N-SDU is used in 1 to 1 communication, thus physical addressing will be supported for all types of network layer messages. In terms of application, physical addressing is used by the external (or internal) tester if it knows the physical address of an ECU that should respond to a service request. When physical addressing is used, a point to point communication takes place (1 to 1 communication). Use cases are (for example) messages, such as ”ReadDataByIdentifier” or ”InputOutputControl ByIdentifier”
Single connection	The CAN Transport Layer will only manage one transport protocol communication session at a time.
Connection channel	The CAN Transport Layer is handling resources used by multiple connections in order to save RAM. When a connection becames active, the channel that is used by this connection will be unavailable for other connections.
Connection	A transport protocol session, either is a transmission or a reception session on a N-SDU.

Table 2.1: Concepts of CAN Transport Layer

3 Related documentation

3.1 Input documents & related standards and norms

[1] Road vehicles – Diagnostics on Controller Area Networks (CAN) – Part2: Network layer services

[2] Diagnostics on controller area network (CAN) – Part 4: Requirements for emission-related systems (Release 2005 01-04)

[3] Glossary
AUTOSAR_TR_Glossary

[4] General Specification of Basic Software Modules
AUTOSAR_SWS_BSWGeneral

[5] Specification of PDU Router
AUTOSAR_SWS_PDURouter

[6] Specification of CAN Interface
AUTOSAR_SWS_CANInterface

[7] General Requirements on Basic Software Modules
AUTOSAR_SRS_BSWGeneral

[8] Requirements on CAN
AUTOSAR_SRS_CAN

3.2 Related specification

AUTOSAR provides a General Specification on Basic Software modules (see [4]), which is also valid for CAN Transport Layer. Thus, the specification SWS BSW General [4] shall be considered as additional and required specification for CAN Transport Layer.

4 Constraints and assumptions

4.1 Limitations

The AUTOSAR architecture defines communication system specific transport layers (CanTp, LinTp including LinIf, FlexRayTp). Thus the CAN Transport Layer only covers CAN transport protocol specifics.

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

According to the AUTOSAR release plan, this CAN Transport Layer specification has the following restriction:

- CAN Transport Layer runs only in an event triggered mode

4.2 Applicability in automotive domain

The CAN Transport Layer can be used for all domains whenever the CAN communication system is connected to the appropriate ECU.

5 Dependencies to other modules

This section sets out relations between the CanTp and other AUTOSAR basic software modules. It contains short descriptions of some AUTOSAR basic concepts, configuration information and services, which are required by the CanTp from other modules.

5.1 AUTOSAR architecture basic concepts

5.1.1 CAN Transport Layer connection(s)

In the AUTOSAR architecture final release, transport protocol facilities will be used to transport both diagnostic (e.g. OBD and UDS protocols) and AUTOSAR COM I-PDUs.

Therefore, the CanTp module is able to deal with multiple connections simultaneously (i.e. multiple segmentation sessions in parallel).

The maximum number of simultaneous 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.

To allow the user to choose which I-PDUs could be received (or sent) simultaneously, each N-SDU identifier will be internally routed through a configured CanTp "connection channel". Since a "connection channel" is not accessible externally, all necessary information (see Chapter 10.2) to transfer an N-SDU will be linked to the N-SDU identifier (e.g. "connection channel" number, timeouts, addressing format, and so on).

Depending on the Meta Data configuration, an N-SDU acts either as a specific connection with defined N_AI, or as a generic connection, where the N_TA, N_SA, and N_AE vary at runtime, while N_TAtype, MType, and the addressing format are statically defined.

5.1.2 CAN Transport Layer interactions

The figure below shows the interactions between CanTp, PduR and CanIf modules.

The CanTp's upper interface offers the PduR module global access, to transmit and receive data. This access is achieved by CAN N-SDU identifier (CAN NSduId). CAN NSduId refers to a constant data structure which consists of attributes describing CAN N-SDU. Each CAN N-SDU specific data structure may contain attributes such as: type of N-SDU (Tx or Rx), its addressing format, L-SDU identifier of this message or other attributes that are useful for implementation.

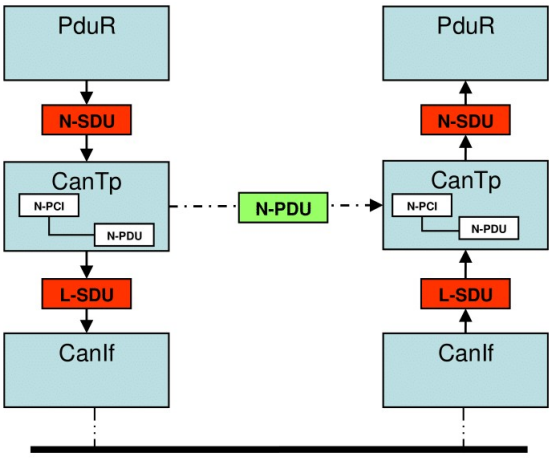

Figure 5.1: CAN Transport Layer interactions

5.1.3 Processing mode

The AUTOSAR communication stack supports both polling and event triggering mode. Therefore, each communication layer can receive information from its lower layer and propagate information to its upper layer by different mechanisms.

In the case of the CAN Transport Layer, only the event triggering mode is supported.

5.1.4 Data consistency

To optimize the communication stack, AUTOSAR limits the CAN Transport Layer buffering capacity. Therefore, the CanTp copies N-SDU payload directly from the upper layer (DCM, COM or PDU Router - in the case of 1:1 TP routing) to the CAN driver and vice-versa. Thus to guarantee data consistency, the upper layer will observe the following rules:

- At transmission time, the N-SDU data payload will remain unchanged, from transmit request until transmit confirmation has been received
- At reception time, the N-SDU data access will be locked, from start of reception until the reception indication has been received.

5.1.5 Static configuration

At runtime the CAN Transport module must have all information required to manage transport connection. Therefore, the following properties should be statically configured:

- Number of CAN N-SDU
- Unique identifier of each CAN N-SDU
- Communication direction of each CAN N-SDU (Tx or Rx)
- Addressing format of each connection (normal, extended, mixed 11bit, normal fixed, or mixed 29 bit) and, depending on the addressing format, additionally:
  - Normal: none
  - Extended: `N_TA`
  - Mixed 11 bit: `N_AE`
  - Normal fixed: `N_TA`, `N_SA`
  - Mixed 29 bit: `N_TA`, `N_SA`, `N_AE`

The static addressing information may be omitted for generic connections that use N-SDUs with `MetaData`.

- Addressing format of each connection (normal, extended or mixed) and, in the case of extended addressing format, the N_TA value or in case of mixed addressing format the N_AE value.
- Associated CAN L-SDU identifier of each CAN N-SDU identifier and if necessary (multiple frame segmentation session) the CAN L-SDU identifier used to transmit the CAN FC N-PDU
- Classic CAN frames and CAN FD frames

The configuration of the CAN Transport Layer can be performed during compilation or post-build (See Chapter 10).

5.1.6 PDU Router services

The CAN Transport Layer uses callback functions of the PDU Router to copy transmit data and to confirm transmission, to initiate reception, copy received data and to indicate reception of a message:

- PduR_CanTpRxIndication
- PduR_CanTpStartOfReception
- PduR_CanTpCopyRxData
- PduR_CanTpCopyTxData
- PduR_CanTpTxConfirmation

For more information about these functions, refer to the PDU Router module specification [5].

5.1.7 CAN Interface services

The CAN Transport Layer uses the following services of the CAN Interface to transmit CAN N-PDUs:

- CanIf_Transmit

For more information about this function, refer to the CAN Interface module specification [6].

5.2 File structure

5.2.1 Code file structure

For details refer to the chapter 5.1.6 “Code file structure” in SWS_BSWGeneral.

5.2.2 Header file structure

AUTOSAR specifies that an ECU can be created from modules provided as object code, source code (generated or not) and even mixed.

The decision to provide a module as object code or source code is based on a compromise between IP protection, test coverage, code efficiency and configurability at system generation time. Thus depending on the configurability requirements of the OEM, suppliers may deliver the CanTp module as object code, generated code or source code.

5.2.3 Version check

[SWS_CanTp_00267] ⌈Version number macros can be used for checking and reading out the software version of a software module, during compile-time and run-time.⌋*()*

5.2.4 Design Rules

For details refer to the chapters 7.1.4, 7.1.8, 7.1.9 in SWS_BSWGeneral.

6 Requirements Tracing

The following tables reference the requirements specified in [7] and [8] and links to the fulfillment of these. Please note that if column “Satisfied by” is empty for a specific requirement this means that this requirement is not fulfilled by this document.

Requirement	Description	Satisfied by
[SRS_BSW_00010]	The memory consumption of all Basic SW Modules shall be documented for a defined configuration for all supported platforms.	[SWS_CanTp_00327]
[SRS_BSW_00101]	The Basic Software Module shall be able to initialize variables and hardware in a separate initialization function	[SWS_CanTp_00208]
[SRS_BSW_00159]	All modules of the AUTOSAR Basic Software shall support a tool based configuration	[SWS_CanTp_00146]
[SRS_BSW_00161]	The AUTOSAR Basic Software shall provide a microcontroller abstraction layer which provides a standardized interface to higher software layers	[SWS_CanTp_00327]

Requirement	Description	Satisfied by
[SRS_BSW_00162]	The AUTOSAR Basic Software shall provide a hardware abstraction layer	[SWS_CanTp_00327]
[SRS_BSW_00167]	All AUTOSAR Basic Software Modules shall provide configuration rules and constraints to enable plausibility checks	[SWS_CanTp_00147]
[SRS_BSW_00168]	SW components shall be tested by a function defined in a common API in the Basis-SW	[SWS_CanTp_00327]
[SRS_BSW_00170]	The AUTOSAR SW Components shall provide information about their dependency from faults, signal qualities, driver demands	[SWS_CanTp_00327]
[SRS_BSW_00172]	The scheduling strategy that is built inside the Basic Software Modules shall be compatible with the strategy used in the system	[SWS_CanTp_00327]
[SRS_BSW_00307]	Global variables naming convention	[SWS_CanTp_00327]
[SRS_BSW_00314]	All internal driver modules shall separate the interrupt frame definition from the service routine	[SWS_CanTp_00327]
[SRS_BSW_00321]	The version numbers of AUTOSAR Basic Software Modules shall be enumerated according specific rules	[SWS_CanTp_00327]
[SRS_BSW_00325]	The runtime of interrupt service routines and functions that are running in interrupt context shall be kept short	[SWS_CanTp_00327]
[SRS_BSW_00328]	All AUTOSAR Basic Software Modules shall avoid the duplication of code	[SWS_CanTp_00327]
[SRS_BSW_00334]	All Basic Software Modules shall provide an XML file that contains the meta data	[SWS_CanTp_00327]
[SRS_BSW_00336]	Basic SW module shall be able to shutdown	[SWS_CanTp_00010]
[SRS_BSW_00339]	Reporting of production relevant error status	[SWS_CanTp_00008]
[SRS_BSW_00341]	Module documentation shall contains all needed informations	[SWS_CanTp_00327]
[SRS_BSW_00342]	It shall be possible to create an AUTOSAR ECU out of modules provided as source code and modules provided as object code, even mixed	[SWS_CanTp_00327]
[SRS_BSW_00344]	BSW Modules shall support link-time configuration	[SWS_CanTp_00327]

Requirement	Description	Satisfied by
[SRS_BSW_00347]	A Naming seperation of different instances of BSW drivers shall be in place	[SWS_CanTp_00327]
[SRS_BSW_00353]	All integer type definitions of target and compiler specific scope shall be placed and organized in a single type header	[SWS_CanTp_00002]
[SRS_BSW_00358]	The return type of init() functions implemented by AUTOSAR Basic Software Modules shall be void	[SWS_CanTp_00208]
[SRS_BSW_00361]	All mappings of not standardized keywords of compiler specific scope shall be placed and organized in a compiler specific type and keyword header	[SWS_CanTp_00327]
[SRS_BSW_00373]	The main processing function of each AUTOSAR Basic Software Module shall be named according the defined convention	[SWS_CanTp_00164]
[SRS_BSW_00375]	Basic Software Modules shall report wake-up reasons	[SWS_CanTp_00327]
[SRS_BSW_00378]	AUTOSAR shall provide a boolean type	[SWS_CanTp_00327]
[SRS_BSW_00383]	The Basic Software Module specifications shall specify which other configuration files from other modules they use at least in the description	[SWS_CanTp_00327]
[SRS_BSW_00397]	The configuration parameters in pre-compile time are fixed before compilation starts	[SWS_CanTp_00327]
[SRS_BSW_00398]	The link-time configuration is achieved on object code basis in the stage after compiling and before linking	[SWS_CanTp_00327]
[SRS_BSW_00399]	Parameter-sets shall be located in a separate segment and shall be loaded after the code	[SWS_CanTp_00327]
[SRS_BSW_00400]	Parameter shall be selected from multiple sets of parameters after code has been loaded and started	[SWS_CanTp_00327]
[SRS_BSW_00404]	BSW Modules shall support post-build configuration	[SWS_CanTp_00327]
[SRS_BSW_00405]	BSW Modules shall support multiple configuration sets	[SWS_CanTp_00327]
[SRS_BSW_00406]	A static status variable denoting if a BSW module is initialized shall be initialized with value 0 before any APIs of the BSW module is called	[SWS_CanTp_00161]

Requirement	Description	Satisfied by
[SRS_BSW_00413]	An index-based accessing of the instances of BSW modules shall be done	[SWS_CanTp_00327]
[SRS_BSW_00414]	Init functions shall have a pointer to a configuration structure as single parameter	[SWS_CanTp_00208]
[SRS_BSW_00415]	Interfaces which are provided exclusively for one module shall be separated into a dedicated header file	[SWS_CanTp_00327]
[SRS_BSW_00416]	The sequence of modules to be initialized shall be configurable	[SWS_CanTp_00327]
[SRS_BSW_00417]	Software which is not part of the SW-C shall report error events only after the DEM is fully operational.	[SWS_CanTp_00327]
[SRS_BSW_00419]	If a pre-compile time configuration parameter is implemented as "const" it should be placed into a separate c-file	[SWS_CanTp_00327]
[SRS_BSW_00422]	Pre-de-bouncing of error status information is done within the DEM	[SWS_CanTp_00327]
[SRS_BSW_00423]	BSW modules with AUTOSAR interfaces shall be describable with the means of the SW-C Template	[SWS_CanTp_00327]
[SRS_BSW_00424]	BSW module main processing functions shall not be allowed to enter a wait state	[SWS_CanTp_00164]
[SRS_BSW_00427]	ISR functions shall be defined and documented in the BSW module description template	[SWS_CanTp_00327]
[SRS_BSW_00428]	A BSW module shall state if its main processing function(s) has to be executed in a specific order or sequence	[SWS_CanTp_00327]
[SRS_BSW_00429]	Access to OS is restricted	[SWS_CanTp_00327]
[SRS_BSW_00432]	Modules should have separate main processing functions for read/receive and write/transmit data path	[SWS_CanTp_00327]
[SRS_BSW_00433]	Main processing functions are only allowed to be called from task bodies provided by the BSW Scheduler	[SWS_CanTp_00327]
[SRS_Can_01065]	The AUTOSAR CAN Transport Layer shall be based on ISO 15765-2 and 15765-4 specifications	[SWS_CanTp_00033] [SWS_CanTp_00350]

Requirement	Description	Satisfied by
[SRS_Can_01066]	The AUTOSAR CAN Transport Layer shall be statically configurable to support either single or multiple connections in an optimizing way	[SWS_CanTp_00096] [SWS_CanTp_00120] [SWS_CanTp_00121] [SWS_CanTp_00122] [SWS_CanTp_00123] [SWS_CanTp_00124]
[SRS_Can_01068]	The CAN Transport Layer shall identify each N-SDU with a unique identifier.	[SWS_CanTp_00035]
[SRS_Can_01069]	CAN address information and N-SDU identifier mapping	[SWS_CanTp_00035]
[SRS_Can_01071]	The CAN Transport Layer shall identify each N-PDU (also called L-SDU) with a unique identifier	[SWS_CanTp_00035] [SWS_CanTp_00231] [SWS_CanTp_00232]
[SRS_Can_01073]	The CAN Transport Layer shall be statically configured to pad unused bytes of PDU	[SWS_CanTp_00116] [SWS_CanTp_00344] [SWS_CanTp_00345] [SWS_CanTp_00346] [SWS_CanTp_00348] [SWS_CanTp_00351]
[SRS_Can_01074]	The Transport connection properties shall be statically configured	[SWS_CanTp_00231] [SWS_CanTp_00232]
[SRS_Can_01075]	The CAN Transport Layer shall implement an interface for initialization	[SWS_CanTp_00030] [SWS_CanTp_00170]
[SRS_Can_01076]	The CAN Transport Layer services shall not be operational before initializing the module	[SWS_CanTp_00031]
[SRS_Can_01078]	The AUTOSAR CAN Transport Layer shall support the ISO 15765-2 addressing formats	[SWS_CanTp_00035]
[SRS_Can_01079]	The CAN Transport Layer shall be compliant with the CAN Interface module notifications	[SWS_CanTp_00086]
[SRS_Can_01082]	Error handling	[SWS_CanTp_00057]
[SRS_Can_01086]	Data padding value of unused bytes	[SWS_CanTp_00059]
[SRS_Can_01163]	The AUTOSAR CAN Transport Layer shall support classic CAN and CAN FD communication as specified by ISO 15765-2	[SWS_CanTp_00354]

7 Functional specification

This section provides a description of the CAN Transport Layer functionality. It explains the services provided to the upper and lower layers and the internal behavior of the CAN Transport Layer.

The CanTp module offers services for segmentation, transmission with flow control, and reassembly of messages. Its main purpose is to transmit and receive messages

that may or may not fit into a single CAN frame. Messages that do not fit into a single CAN frame are segmented into multiple parts, such that each can be transmitted in a single CAN frame.

While reading this document, it is necessary to bear in mind, that this module will follow the recommendations ISO 15765-2 (OEM enhanced diagnostics [1]) and should be able to fulfill ISO 15765-4 (Requirements for emissions-related systems [2]).

[SWS_CanTp_00033] ⌈If a recommendation of ISO 15765-2 is not explicitly excluded in the SWS, the CanTp module shall follow this recommendation.⌋*(SRS_Can_01065)*

For further descriptions of SF, FF, CF and FC frames, network layer timing parameters, and further functionalities of CAN Transport Layer please refer to the ISO 15765-2 specification [1].

ISO 15765-4 is a particular case of ISO-15765-2. Therefore, the CAN Transport Layer will be configurable in order to be able to adapt the module to all ISO 15765-4 use cases (e.g. specific timing, padding configuration, addressing mode). See chapter 10, Configuration specification, for details.

7.1 Services provided to upper layer

The service interface of the CanTp module can be divided into the following main categories:

- Initialization and shutdown
- Communication services

The following paragraphs describe the functionality of each services category.

7.1.1 Initialization and shutdown

[SWS_CanTp_00027] ⌈The CanTp module shall have two internal states, `CANTP_OFF` and `CANTP_ON`.⌋*()*

[SWS_CanTp_00168] ⌈The CanTp module shall be in the `CANTP_OFF` state after power up.⌋*()*

[SWS_CanTp_00169] ⌈In the state `CANTP_OFF`, the CanTp shall allow an update of the postbuild configuration.⌋*()*

[SWS_CanTp_00170] ⌈The CanTp module shall change to the internal state `CANTP_ON` when the CanTp has been successfully initialized with `CanTp_Init`().⌋*(SRS_Can_01075)*

[SWS_CanTp_00238] ⌈The CanTp module shall perform segmentation and reassembly tasks only when the CanTp is in the `CANTP_ON` state.⌋*()*

[SWS_CanTp_00030] ⌈The function `CanTp_Init` shall initialize all global variables of the module and sets all transport protocol connections in a sub-state of `CANTP_ON`, in which neither segmented transmission nor segmented reception are in progress (Rx thread in state `CANTP_RX_WAIT` and Tx thread in state `CANTP_TX_WAIT`).⌋*(SRS_Can_01075)*

[SWS_CanTp_00031] ⌈If development error detection for the CanTp module is enabled the CanTp module shall raise an error (`CanTp.CANTP_E_UNINIT`) when the PDU Router or CAN Interface tries to use any function (except `CanTp_GetVersionInfo`) before the function `CanTp_Init` has been called.⌋*(SRS_Can_01076)*

[SWS_CanTp_00111] ⌈If called when the CanTp module is in the global state `CANTP_ON`, the function `CanTp_Init` shall return the module to state Idle (state = `CANTP_ON`, but neither transmission nor reception are in progress).⌋*()*

[SWS_CanTp_00273] ⌈The CanTp module shall loose all current connections if `CanTp_Init` is called when CanTp module is in the global state `CANTP_ON`.⌋*()*

[SWS_CanTp_00010] ⌈The function `CanTp_Shutdown` shall stop the CanTp module properly.⌋*(SRS_BSW_00336)*

The following figure summarizes all of the above requirements:

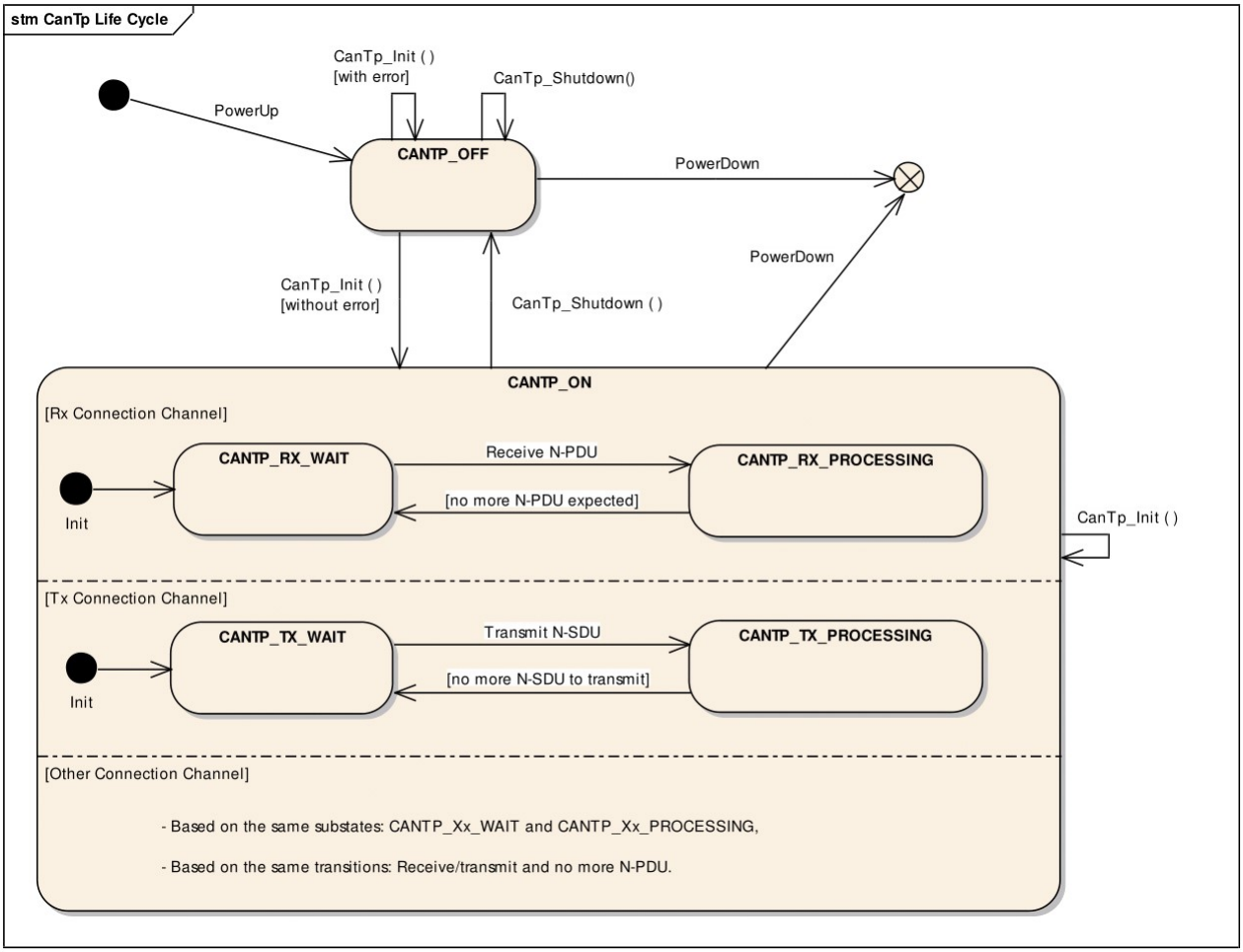

Figure 7.1: CAN Transport Layer life cycle

7.1.2 Transmit request

The transmit operation, CanTp_Transmit(), will allow upper layers to ask for data transfer using CAN transport protocol facilities (segmentation, extended addressing format and so on).

[SWS_CanTp_00176] ⌈The function CanTp_Transmit() shall be asynchronous.⌋*()*

[SWS_CanTp_00177] ⌈After the transmit request was accepted, the CanTp module shall notify its upper layer if the N-SDU transfer is fully processed (successfully or not).⌋*()*

7.1.3 Transmit cancellation

The transmit cancellation feature allows the upper layer to cancel a transmission in progress.

Use case: Cancel a diagnostic transmission due to the reception of another diagnostic protocol with higher priority.

[SWS_CanTp_00242] ⌈This feature shall be (de)activated by static configuration (parameter CanTpTc).⌋*()*

[SWS_CanTp_00274] ⌈Transmit Cancellation is triggered by the call of CanTp_CancelTransmit().⌋*()*

[SWS_CanTp_00243] ⌈After the call of the service CanTp_CancelTransmit(), the transfer on this connection shall be aborted.⌋*()*

Note: The Api PduR_CanTpTxConfirmation() shall be called after a transmit cancellation with value E_NOT_OK. (see also [SWS_CanTp_00255])

Note that if a transfer is in progress, that will generate a time-out error on the receiver side.

7.2 Services provided to the lower layer

According to the AUTOSAR specification of the communication stack, the CAN Transport Layer provides the following two callback functions to the Can interface: CanTp_TxConfirmation() and CanTp_RxIndication().

7.2.1 Transmit confirmation

The CanIf module shall call the transmit confirmation function to notify the CAN Transport Layer that a CAN frame transmission, requested by the CanTp, has been per-

formed successfully or not. The L-PDU identifier is associated with the call in order to identify the corresponding transmission.

[SWS_CanTp_00075] ⌈When the transmit confirmation is not received after a maximum time (equal to N_As), the CanTp module shall abort the corresponding session. The N-PDU remains unavailable to other concurrent sessions until the TxConfirmation is received, successful or not.⌋*()*

[SWS_CanTp_00076] ⌈For confirmation calls, the CanTp module shall provide the function CanTp_TxConfirmation().⌋*()*

[SWS_CanTp_00355] ⌈CanTp shall abort the correnspondingsession, when CanTp_TxConfirmation() is called with the result E_NOT_OK.⌋*()*

7.2.2 Reception indication

The CanIf module shall call the reception indication function to notify the CanTp module that a new CAN N-PDU frame (i.e. a transport protocol frame) has been received.

The reception indication can be performed in ISR context according to CanIf configuration.

[SWS_CanTp_00078] ⌈For reception indication, the CanTp module shall provide CanTp_RxIndication().⌋*()*

7.3 Internal behavior

The internal operation of the CAN Transport Layer provides basic mechanisms in order to perform the main purpose of this module, which is to transfer messages in a single CAN frame or in multiple CAN frames.

The entire behavior of the CAN Transport Layer will be event triggered, so that CanTp can processes transfer of N-SDU (respectively L-SDU) coming from the PDU Router (respectively CAN Interface) directly.

7.3.1 N-SDU Reception

[SWS_CanTp_00079] ⌈When receiving an SF or an FF N-PDU, the CanTp module shall notify the upper layer (PDU Router) about this reception using the PduR_CanTpStartOfReception function.⌋*()*

Note: The upper layer will reserve and lock a buffer for reception of the N-SDU.

[SWS_CanTp_00329] ⌈CanTp shall provide the content of the FF/SF to PduR using the parameter TpSduInfoPtr of PduR_CanTpStartOfReception().⌋*()*

[SWS_CanTp_00350] ⌈The received data link layer data length (RX_DL) shall be derived from the first received payload length of the CAN frame/PDU (CAN_DL) as follows:

- For CAN_DL values less than or equal to eight bytes the RX_DL value shall be eight.
- For CAN_DL values greater than eight bytes the RX_DL value equals the CAN_DL value.

⌋(*SRS_Can_01065*)

Note: ISO frame overview table:

N_PDU name	N_PCI bytes						
	Byte #1		Byte #2	Byte #3	Byte #4	Byte #5	Byte #6
	Bits 7 – 4	Bits 3 – 0					
SingleFrame (SF) (CAN_DL ≤ 8) [a]	0000	SF_DL					
SingleFrame (SF) (CAN_DL > 8) [b]	0000	0000	SF_DL				
FirstFrame (FF) (FF_DL <= 4095) [a]	0001	FF_DL					
FirstFrame (FF) (FF_DL > 4095) [a]	0001	0000	0000 0000	FF_DL			
ConsecutiveFrame (CF) [a]	0010	SN					
FlowControl (FC) [a]	0011	FS	BS	ST_{min}	N/A	N/A	N/A
NOTE Shaded cells are not utilized for PCI information, but depending on the PDU, they might be utilized for payload data.							
[a] CAN 2.0 or CAN FD [b] CAN FD only							

Figure 7.2: Summary of N_PCI bytes

[SWS_CanTp_00330] ⌈When CanTp_RxIndication is called for a SF or FF N-PDU with MetaData (indicating a generic connection), the CanTp module shall store the addressing information contained in the MetaData of the PDU and use this information for the initiation of the connection to the upper layer, for transmission of FC N-PDUs and for identification of CF N-PDUs. The addressing information in the MetaData depends on the addressing format:

- Normal, Extended, Mixed 11 bit: none
- Normal fixed, Mixed 29 bit: N_SA, N_TA

⌋()

[SWS_CanTp_00331] ⌈When calling PduR_CanTpStartOfReception() for a generic connection (N-SDU with MetaData), the CanTp module shall forward the

extracted addressing information via the MetaData of the N-SDU. The addressing information in the MetaData depends on the addressing format:

- Normal: none
- Extended: N_TA
- Mixed 11 bit: N_AE
- Normal fixed: N_SA, N_TA
- Mixed 29 bit: N_SA, N_TA, N_AE

⌋()

[SWS_CanTp_00332] ⌈When calling CanIf_Transmit() for an FC on a generic connection (N-PDU with MetaData), the CanTp module shall provide the stored addressing information via the MetaData of the N-PDU. The addressing information in the MetaData depends on the addressing format:

- Normal, Extended, Mixed 11 bit: none
- Normal fixed, Mixed 29 bit: N_SA (saved N_TA), N_TA (saved N_SA)

⌋()

[SWS_CanTp_00333] ⌈When CanTp_RxIndication is called for a CF on a generic connection (N-PDU with MetaData), the CanTp module shall check the addressing information contained in the MetaData of the N-PDU against the stored values from the FF.⌋()

[SWS_CanTp_00166] ⌈At the reception of a FF or last CF of a block, the CanTp module shall start a time-out N_Br before calling PduR_CanTpStartOfReception or PduR_CanTpCopyRxData.⌋()

[SWS_CanTp_00080] ⌈The available Rx buffer size is reported to the CanTp in the output pointer parameter of the PduR_CanTpStartOfReception() service. The available Rx buffer can be smaller than the expected N-SDU data length.⌋()

Note: If the upper layer cannot make a buffer available because of an error (e.g. in the gateway case it may indicate that the transport session to the destination network has been broken) or a resource limitation (e.g. N-SDU length exceeds the maximum buffer size of the upper layer), the PduR_CanTpStartOfReception() function returns BUFREQ_E_NOT_OK or BUFREQ_E_OVFL.

[SWS_CanTp_00081] ⌈After the reception of a First Frame or Single Frame, if the function PduR_CanTpStartOfReception() returns BUFREQ_E_NOT_OK to the Can Tp module, the CanTp module shall abort the reception of this N-SDU. No Flow Control will be sent and PduR_CanTpRxIndication() will not be called in this case.⌋()

[SWS_CanTp_00318] ⌈After the reception of a First Frame, if the function PduR_CanTpStartOfReception() returns BUFREQ_E_OVFL to the CanTp module, the Can

Tp module shall send a Flow Control N-PDU with overflow status (FC(OVFLW)) and abort the N-SDU reception.⌋()

[SWS_CanTp_00353] ⌈After the reception of a Single Frame, if the function PduR_CanTpStartOfReception() returns BUFREQ_E_OVFL to the CanTp module, the Can Tp module shall abort the N-SDU reception.⌋()

[SWS_CanTp_00339] ⌈After the reception of a First Frame or Single Frame, if the function PduR_CanTpStartOfReception() returns BUFREQ_OK with a smaller available buffer size than needed for the already received data, the CanTp module shall abort the reception of the N-SDU and call PduR_CanTpRxIndication() with the result E_NOT_OK.⌋()

[SWS_CanTp_00082] ⌈After the reception of a First Frame, if the function PduR_CanTpStartOfReception() returns BUFREQ_OK with a smaller available buffer size than needed for the next block, the CanTp module shall start the timer N_Br.⌋()

[SWS_CanTp_00325] ⌈If the function PduR_CanTpCopyRxData() called after reception of the last Consecutive Frame of a block returns BUFREQ_OK, but the remaining buffer is not sufficient for the reception of the next block, the CanTp module shall start the timer N_Br.⌋()

[SWS_CanTp_00222] ⌈While the timer N_Br is active, the CanTp module shall call the service PduR_CanTpCopyRxData() with a data length 0 (zero) and NULL_PTR as data buffer during each processing of the MainFunction.⌋()

Note: ISO 15765-2 specification defines the following performance requirement: (N_Br + N_Ar) < 0.9 * N_Bs timeout.

[SWS_CanTp_00341] ⌈If the N_Br timer expires and the available buffer size is still not big enough, the CanTp module shall send a new FC(WAIT) to suspend the N-SDU reception and reload the N_Br timer.⌋()

[SWS_CanTp_00223] ⌈The CanTp module shall send a maximum of WFTmax consecutive FC(WAIT) N-PDU. If this number is reached, the CanTp module shall abort the reception of this N-SDU (the receiver did not send any FC N-PDU, so the N_Bs timer expires on the sender side and then the transmission is aborted) and a receiving indication with E_NOT_OK occurs.⌋()

[SWS_CanTp_00311] ⌈In case of N_Ar timeout occurrence (no confirmation from CAN driver for any of the FC frame sent) the CanTp module shall abort reception and notify the upper layer of this failure by calling the indication function PduR_CanTpRxIndication() with the result E_NOT_OK.⌋()

[SWS_CanTp_00224] ⌈When the Rx buffer is large enough for the next block (directly after the First Frame or the last Consecutive Frame of a block, or after repeated calls to PduR_CanTpCopyRxData() according to [SWS_CanTp_00222]), the CanTp module shall send a Flow Control N-PDU with ClearToSend status (FC(CTS)) and shall then expect the reception of Consecutive Frame N-PDUs.⌋()

[SWS_CanTp_00269] ⌈After reception of each Consecutive Frame the CanTp module shall call the PduR_CanTpCopyRxData() function with a PduInfo pointer containing data buffer and data length:

- 6 or 7 bytes or less in case of the last CF for CAN 2.0 frames

- DLC-1 or DLC-2 bytes for CAN FD frames (see [SWS_CanTp_00351]).

The output pointer parameter provides CanTp with available Rx buffer size after data have been copied.⌋*()*

Note: For details refer to Figure 7.3.

[SWS_CanTp_00312] ⌈The CanTp module shall start a time-out N_Cr at each indication of CF reception (except the last one in a block) and at each confirmation of a FC transmission that initiate a CF transmission on the sender side (FC with FS=CTS).⌋*()*

[SWS_CanTp_00313] ⌈In case of N_Cr timeout occurrence the CanTp module shall abort reception and notify the upper layer of this failure by calling the indication function PduR_CanTpRxIndication() with the result E_NOT_OK.⌋*()*

[SWS_CanTp_00271] ⌈If the PduR_CanTpCopyRxData() returns BUFREQ_E_NOT_OK after reception of a Consecutive Frame in a block the CanTp shall abort the reception of N-SDU and notify the PduR module by calling the PduR_CanTpRxIndication() with the result E_NOT_OK.⌋*()*

[SWS_CanTp_00314] ⌈The CanTp shall check the correctness of each SN received during a segmented reception. In case of wrong SN received the CanTp module shall abort reception and notify the upper layer of this failure by calling the indication function PduR_CanTpRxIndication() with the result E_NOT_OK.⌋*()*

[SWS_CanTp_00084] ⌈When the transport reception session is completed (successfully or not) the CanTp module shall call the upper layer notification service PduR_CanTpRxIndication().⌋*()*

[SWS_CanTp_00277] ⌈With regard to FF N-PDU reception, the content of the Flow Control N-PDU depends on the PduR_CanTpStartOfReception() service result.⌋*()*

[SWS_CanTp_00064] ⌈Furthermore, it should be noted that when receiving a FF N-PDU, the Flow Control shall only be sent after having the result of the PduR_CanTpStartOfReception() service.⌋*()*

[SWS_CanTp_00278] ⌈It is important to note that FC N-PDU will only be sent after every block, composed of a number BS (Block Size) of consecutive frames.⌋*()*

[SWS_CanTp_00067] ⌈The CanTp shall use the same value for the BS and STmin parameters on each FC sent during a segmented reception. Different values of these parameters can be used on different N-SDUs reception.⌋*()*

[SWS_CanTp_00342] ⌈CanTp shall terminate the current reception connection when CanIf_Transmit() returns E_NOT_OK when transmitting an FC.⌋*()*

7.3.2 N-SDU Transmission

As described in chapter 7.1.2, the upper layer asks for the transmission of a N-SDU by calling CanTp_Transmit(). The parameters of CanTp_Transmit() describe the CAN NSduId and the full Tx N-SDU length to be sent .

[SWS_CanTp_00225] ⌈For specific connections that do not use MetaData, the function CanTp_Transmit shall only use the full SduLength information and shall not use the available N-SDU data buffer in order to prepare Single Frame or First Frame PCI.⌋*()*

[SWS_CanTp_00334] ⌈When CanTp_Transmit is called for an N-SDU with MetaData, the CanTp module shall store the addressing information contained in the MetaData of the N-SDU and use this information for transmission of SF, FF, and CF N-PDUs and for identification of FC N-PDUs. The addressing information in the MetaData depends on the addressing format:

- Normal: none
- Extended: N_TA
- Mixed 11 bit: N_AE
- Normal fixed: N_SA, N_TA
- Mixed 29 bit: N_SA, N_TA, N_AE.

⌋*()*

[SWS_CanTp_00335] ⌈When calling CanIf_Transmit() for an SF, FF, or CF of a generic connection (N-PDU with MetaData), the CanTp module shall provide the stored addressing information via MetaData of the N-PDU. The addressing information in the MetaData depends on the addressing format:

- Normal, Extended, Mixed 11 bit: none
- Normal fixed, Mixed 29 bit: N_SA, N_TA.

⌋*()*

[SWS_CanTp_00336] ⌈When CanTp_RxIndication is called for an FC on a generic connection (N-PDU with MetaData), the CanTp module shall check the addressing information contained in the MetaData against the stored values.⌋*()*

[SWS_CanTp_00167] ⌈After a transmission request from upper layer, the CanTp module shall start time-out N_Cs before the call of PduR_CanTpCopyTxData. If no data is available before the timer elapsed, the CanTp module shall abort the communication.⌋*()*

[SWS_CanTp_00086] ⌈The CanTp module shall call the PduR_CanTpCopyTxData service for each segment that is sent (SF, FF and CF). The upper layer copy the transmit data on the PduInfoType structure.⌋*(SRS_Can_01079)*

[SWS_CanTp_00272] ⌈The API PduR_CanTpCopyTxData() contains a parameter used for the recovery mechanism - 'retry'. Because ISO 15765-2 does not support such a mechanism, the CAN Transport Layer does not implement any kind of recovery. Thus, the parameter is always set to NULL pointer.⌋*()*

If the upper layer cannot make Tx data available because of an error (e.g. in gateway case it may indicate that the transport session from the source network was terminated), the PduR_CanTpCopyTxData() function returns BUFREQ_E_NOT_OK.

[SWS_CanTp_00087] ⌈If PduR_CanTpCopyTxData() returns BUFREQ_E_NOT_OK, the CanTp module shall abort the transmit request and notify the upper layer of this failure by calling the callback function PduR_CanTpTxConfirmation() with the result E_NOT_OK.⌋*()*

Note: If upper layer temporarily has no Tx buffer available, the PduR_CanTpCopyTxData() function returns BUFREQ_E_BUSY.

[SWS_CanTp_00184] ⌈If the PduR_CanTpCopyTxData() function returns BUFREQ_E_BUSY, the CanTp module shall later (implementation specific) retry to copy the data.⌋*()*

Note: If no data is available before the expiration of the N_Cs timer (ISO 15765-2 specification defines the following performance requirement: (N_Cs + N_As) < 0.9* N_Cr timeout), the CanTp module shall abort this transmission session.

[SWS_CanTp_00280] ⌈If data is not available within N_Cs timeout the CanTp module shall notify the upper layer of this failure by calling the callback function PduR_CanTpTxConfirmation with the result E_NOT_OK.⌋*()*

[SWS_CanTp_00089] ⌈When Tx data is available, the CanTp module shall resume the transmission of the N-SDU.⌋*()*

[SWS_CanTp_00310] ⌈In case of N_As timeout occurrence (no confirmation from CAN driver) the CanTp module shall notify the upper layer by calling the callback function PduR_CanTpTxConfirmation() with the result E_NOT_OK.⌋*()*

[SWS_CanTp_00309] ⌈If a FC frame is received with the FS set to OVFLW the Can Tp module shall abort the transmit request and notify the upper layer by calling the callback function PduR_CanTpTxConfirmation() with the result E_NOT_OK.⌋*()*

[SWS_CanTp_00317] ⌈If a FC frame is received with an invalid FS the CanTp module shall abort the transmission of this message and notify the upper layer by calling the callback function PduR_CanTpTxConfirmation() with the result E_NOT_OK.⌋*()*

[SWS_CanTp_00315] ⌈The CanTp module shall start a timeout observation for N_Bs time at confirmation of the FF transmission, last CF of a block transmission and at each indication of FC with FS=WT (i.e. time until reception of the next FC).⌋*()*

[SWS_CanTp_00316] ⌈In case of N_Bs timeout occurrence the CanTp module shall abort transmission of this message and notify the upper layer by calling the callback function PduR_CanTpTxConfirmation() with the result E_NOT_OK.⌋*()*

[SWS_CanTp_00090] ⌈When the transport transmission session is successfully completed, the CanTp module shall call a notification service of the upper layer, PduR_CanTpTxConfirmation(), with the result E_OK.⌋*()*

[SWS_CanTp_00343] ⌈CanTp shall terminate the current transmission connection when CanIf_Transmit() returns E_NOT_OK when transmitting an SF, FF, of CF.⌋*()*

7.3.3 Buffer strategy

Because CanTp has no buffering capability, the N-SDU payload, which is to be transmitted, is not copied internally and the N-PDU received is not reassembled internally.

The CAN Transport Layer works directly on the memory area of the upper layers (e.g. PduR, DCM, or COM). To access these memory areas, the CAN Transport Layer uses the PduR_CanTpCopyTxData() or PduR_CanTpCopyRxData() functions.

Thus, to guarantee data consistency, the upper layer should lock this memory area until an indication occurs.

When a transmit buffer is locked, the upper layer must not write data inside the buffer area.

When a receiving buffer is locked the CAN Transport Layer does not guarantee data consistency of the buffer. The upper layer should neither read nor write data in the buffer area.

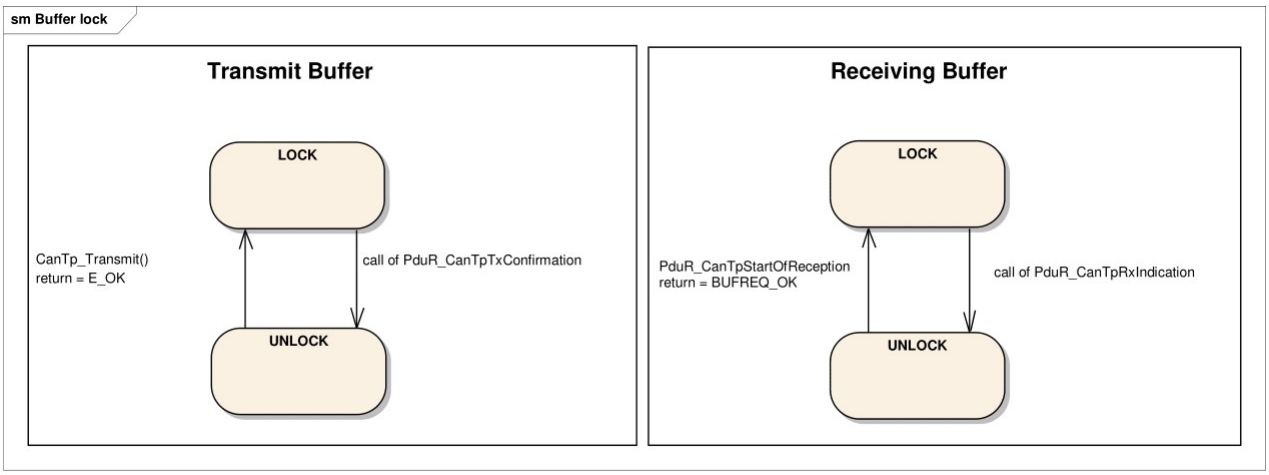

Figure 7.3: Tx and Rx Buffer locking

It is assumed that upper layer module has locked the buffer when it returns a status BUFREQ_OK to a PduR_CanTpStartOfReception() call or when CanTp_Transmit() returns E_OK and shall keep the buffer locked until a confirmation or indication (PduR_CanTpTxConfirmation() or PduR_CanTpRxIndication() call) occurs.

The following figure provides an example, to summarize the process of sending a frame, with a length of 50 bytes utilizing CAN 2.0 frames.

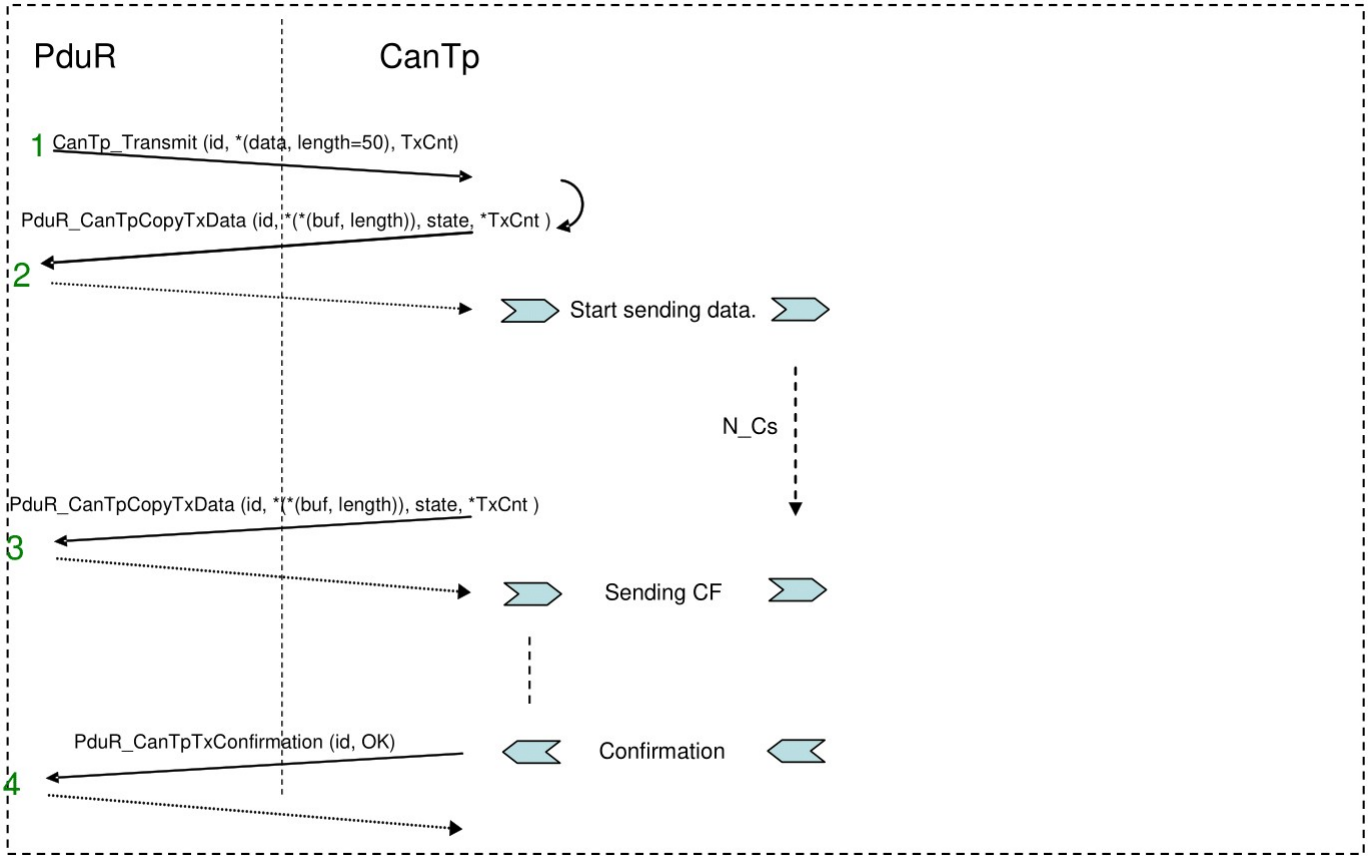

Figure 7.4: Example of transmit process

1: The PduR asks for the transmission of 50 data bytes;

2: The CanTp asks the PduR for the payload data; the CanTp send the First Frame;

3: The CanTp send the rest of payload data as sequences of Consecutive Frames; 6 or 7 payload data bytes are copied by the upper layer on each CF;

4: The CanTp confirms transmission of the payload data.

The next figure is an example of an N-SDU receiving 49 bytes; the upper layer reports 25 bytes as available Rx buffer.

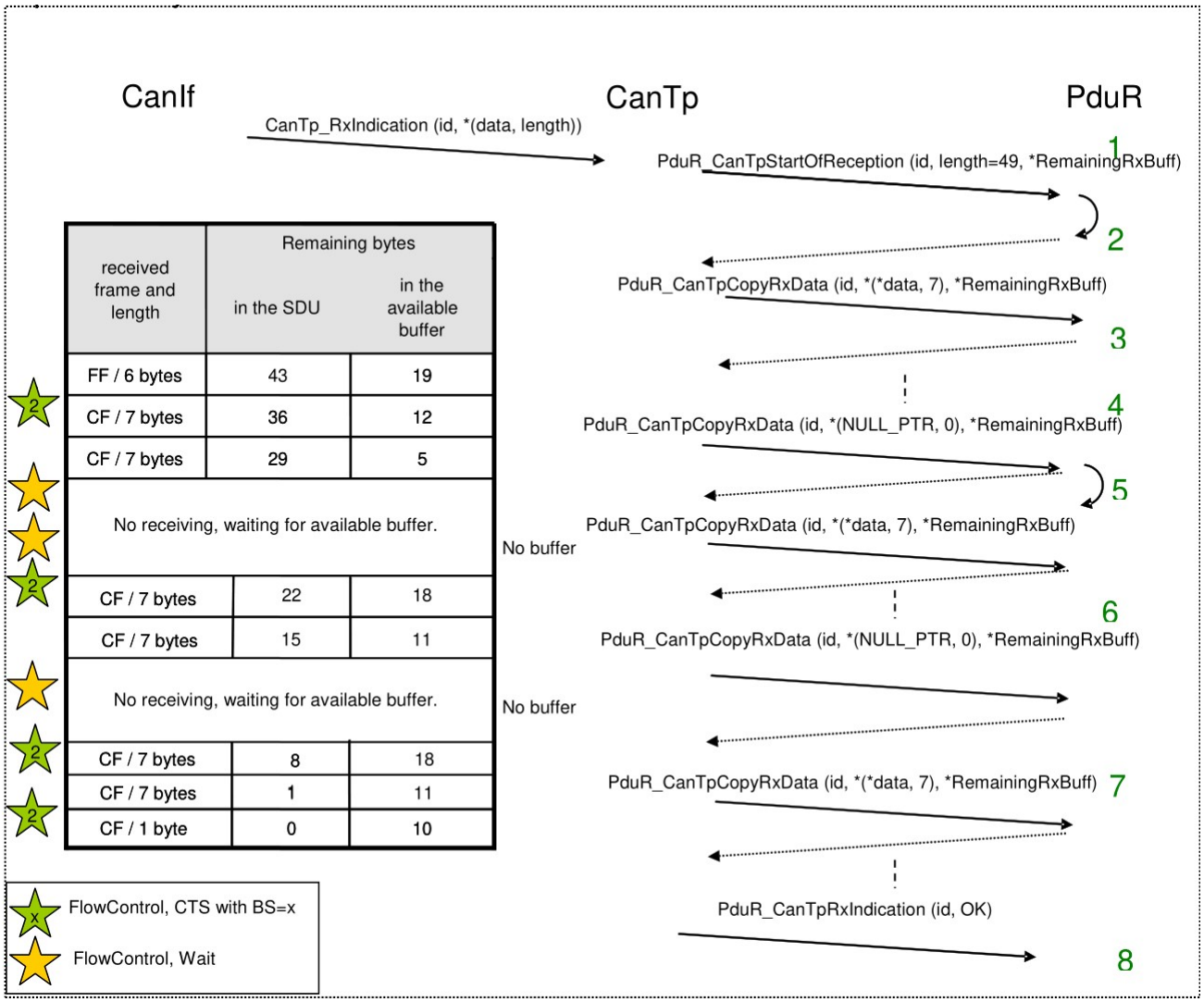

Figure 7.5: Example of receiving process

1: The CanIf notifies a new reception with `CanTp_RxIndication`(). The CanTp forwards this notification to the PduR;

2: The PduR returns an available buffer size of 25 bytes, CanTp sends a FlowControl CTS to the originator;

3: The CanTp provides the data of each received frame to the PduR and monitors the remaining buffer size. After the second consecutive frame, the remaining buffer size is not enough for the next block (two Consecutive Frames);

4: The CanTp asks the PduR for the remaining buffer size by calling `PduR_CanTpCopyRxData`() with 0 as data length and `NULL_PTR` as data, and sends a FlowControl Wait to the originator. This is done until sufficient buffer for the next block is available;

5: When the buffer size is finally sufficient for the next block, the CanTp will send a FlowControl CTS to the originator and continues the reception of the next Consecutive Frames block;

6: After copying the last consecutive frame of the block, the remaining buffer is too low for the next block, so CanTp again sends wait frames and monitors the remaining buffer size;

7: When the buffer for the last block is available, CanTp will continue the reception;

8: The CanTp informs the PduR of the end of reception by a call to `PduR_CanTpRxIndication`().

7.3.4 Protocol parameter setting services

[SWS_CanTp_00091] ⌈The CanTp module shall support optional primitives (proposed in ISO 15765-2 specification) for the dynamic setting of some transport protocol internal parameters (`STmin` and `BS`) by application.

The `BS` value is only a maximum value. For reasons of buffer length, the CAN Transport Layer can adapt the `BS` value within the limit of the configured maximum value.⌋*()*

7.3.5 Tx and Rx data flow

The following figures show examples of an un-segmented message transmission and a segmented one.

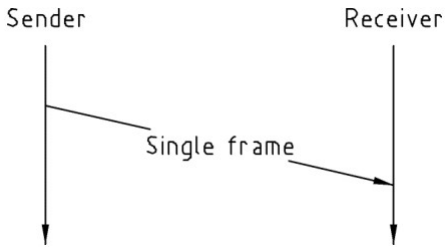

Figure 7.6: Example of single part message

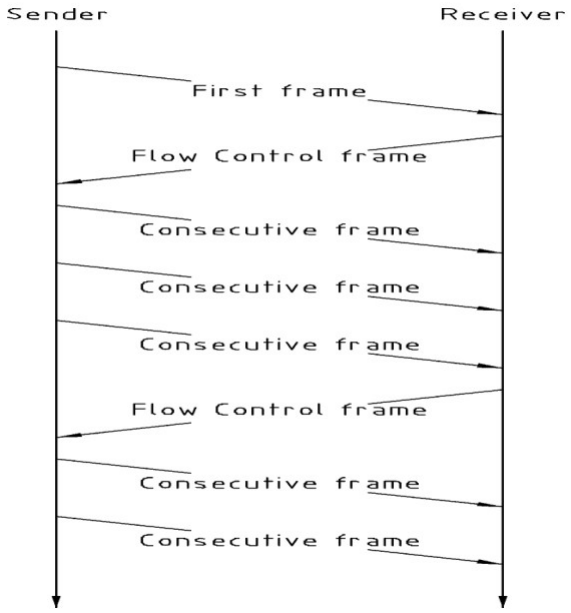

Figure 7.7: Example of multiple parts message

Flow control is used to adjust the sender to the capabilities of the receiver. The main usage of this transport protocol is peer-to-peer communication (i.e. 1 to 1 communication - physical addressing [1]).

[SWS_CanTp_00092] ⌈The CanTp module shall provide 1 to n communication (i.e. functional addressing [1]), in the form of functionality to SF N-PDUs (and only SF N-SDU).⌋*()*

The configuration tool shall check whether it is only SF N-PDUs that have been configured with a functional addressing property.

[SWS_CanTp_00093] ⌈If a multiple segmented session occurs (on both receiver and sender side) with a handle whose communication type is functional, the CanTp module shall reject the request and report the runtime error code `CanTp.CANTP_E_INVALID_TATYPE` to the Default Error Tracer.⌋*()*

7.3.6 Relationship between CAN NSduId and CAN LSduId

This chapter describes the connection that exists between CAN NSduId and CAN LSdu Id, in order to make transmission and reception of transport protocol data units possible.

[SWS_CanTp_00035] ⌈A CAN NSduId shall only be linked to one CAN LSduId that is used to transmit SF, FF, FC and CF frames.⌋*(SRS_Can_01068, SRS_Can_01069, SRS_Can_01071, SRS_Can_01078)*

[SWS_CanTp_00281] ⌈However, if the message is configured to use an extended or a mixed addressing format, the CanTp module must fill the first byte of each transmitted segment (SF, FF and CF) with the `N_TA` (in case of extended addressing) or `N_AE` (in

case of mixed addressing) value. Therefore a CAN NSduId may also be related to a N_TA or N_AE value.⌋()

[SWS_CanTp_00282] ⌈FC protocol data units give receivers the possibility of controlling senders' data flow by authorizing or delaying transmission of subsequent CF N-PDUs.⌋()

[SWS_CanTp_00283] ⌈For extended addressing format, the first data byte of the FC also contains the N_TA value or a unique combination of N_TA and N_TAtype value. For mixed addressing format, the first data byte of the FC contains the N_AE value.⌋()

[SWS_CanTp_00094] ⌈Thus the CAN LSduId of a FC frame combined with its N_TA value (e.g. the N_AI) or with N_AE value shall only identify one CAN NSduId.⌋()

[SWS_CanTp_00284] ⌈In the reception direction, the first data byte value of each (SF, FF or CF) transport protocol data unit will be used to determine the relevant N-SDU.⌋()

[SWS_CanTp_00095] ⌈Therefore, in extended addressing N-PDU reception, the Can Tp module shall extract the N_TA value to establish the related N-SDU. The same process shall be applied for mixed addressing mode in relation with N_AE value.⌋()

The following figure summarizes these discussions.

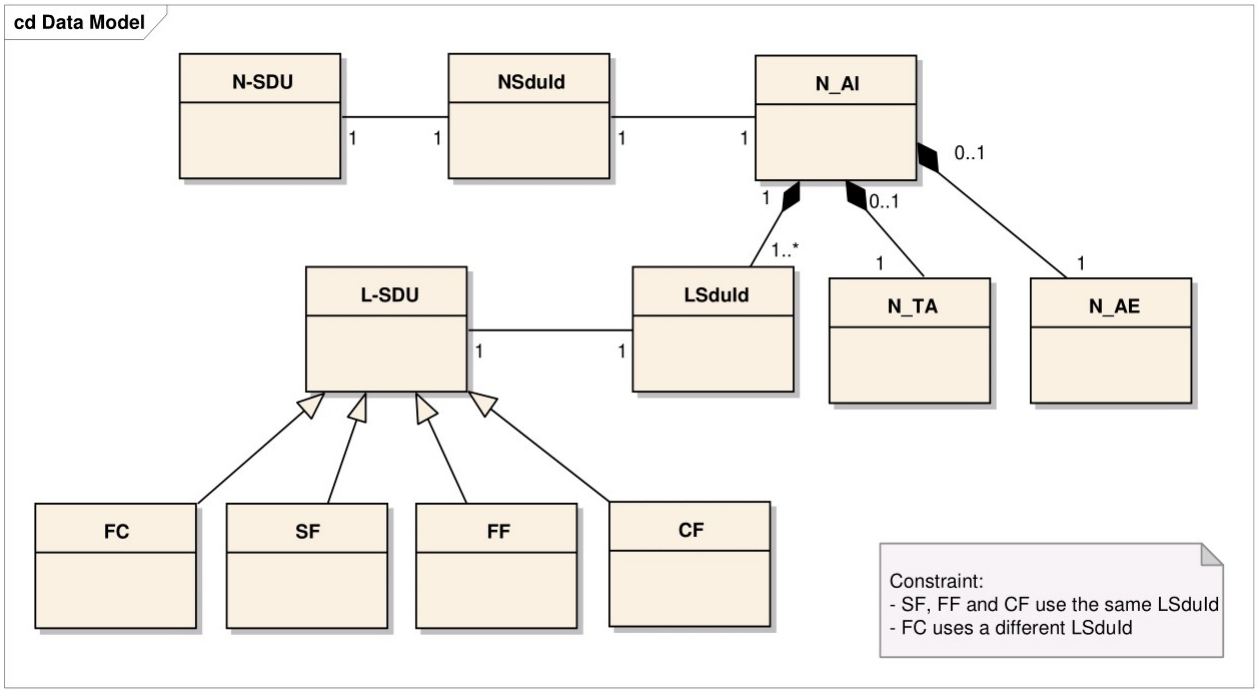

Figure 7.8: Possible links between NSduId and LSduId

7.3.7 Concurrent connection

The CAN Transport Layer is able to manage several connections simultaneously (e.g. a UDS and an OBD request can be received at the same time).

[SWS_CanTp_00096] ⌈The CanTp module shall support several connections simultaneously.⌋*(SRS_Can_01066)*

[SWS_CanTp_00120] ⌈It shall be possible to configure concurrent connections in the CanTp module.⌋*(SRS_Can_01066)*

[SWS_CanTp_00285] ⌈The connection channels are only destined for CAN TP internal use, so they are not accessible externally.⌋*()*

[SWS_CanTp_00286] ⌈All the necessary information (Channel number, Timing parameter . . .) is configured inside the CAN Transport Layer module.⌋*()*

[SWS_CanTp_00121] ⌈Each N-SDU is statically linked to one connection channel. This connection channel represents an internal path, for the transmission or receiving of the N-SDU. A connection channel is attached to one or more N-SDU.⌋*(SRS_Can_01066)*

[SWS_CanTp_00122] ⌈Each connection channel is independent of the other connection channels. This means that a connection channel uses its own resources, such as internal buffer, timer, or state machine.⌋*(SRS_Can_01066)*

[SWS_CanTp_00190] ⌈The CanTp module shall route the N-SDU through the correctly configured connection channel.⌋*()*

[SWS_CanTp_00287] ⌈The CanTp module shall not accept the receiving or the transmission of N-SDU with the same identifier in parallel, because otherwise the received frames cannot be assigned to the correct connection. When only specific connections (without `MetaData`) are used, this requirement is enforced by the configuration, because each N-SDU is linked to only one connection channel.⌋*()*

If a user wants to dedicate a specific connection channel to only one N-SDU, they should assign this connection channel to one N-SDU only during the configuration process.

[SWS_CanTp_00288] ⌈If a connection channel is assigned to multiple N-SDUs, then resources are shared between different N-SDUs, and the CAN Transport Layer will reject transmission or abort receiving, if no free connection channels are available.⌋*()*

[SWS_CanTp_00289] ⌈The number of connection channels is not directly configurable. It will be determined by the configuration tools during the configuration process, by analyzing the N-SDU/Channel routing table.⌋*()*

[SWS_CanTp_00123] ⌈If the configured transmit connection channel is in use (state `CANTP_TX_PROCESSING`), the CanTp module shall reject new transmission requests linked to this channel. To reject a transmission, CanTp returns `E_NOT_OK` when the upper layer asks for a transmission with the `CanTp_Transmit`() function.⌋*(SRS_Can_01066)*

[SWS_CanTp_00124] ⌈When an SF or FF N-PDU without `MetaData` is received, and the corresponding connection channel is currently receiving the same connection (state `CANTP_RX_PROCESSING`, same `N_AI`), the CanTp module shall abort the reception in progress and shall process the received frame as the start of a new reception.

When an SF or FF N-PDU without MetaData is received for another connection (different N_AI) on an active connection channel, the SF or FF shall be ignored.⌋(*SRS_Can_01066*)

[SWS_CanTp_00337] ⌈When an SF or FF N-PDU with MetaData (indicating a generic connection) is received, and the corresponding connection channel is currently receiving, the SF or FF shall be ignored.⌋()

[SWS_CanTp_00248] ⌈When a Tx N-PDU is used by two or more different connections on different channels, access to this N-PDU shall be serialized by using the TxConfirmation. An Rx N-PDU can only be used on two or more different connection channels if extended or mixed addressing is used in relation with this N-PDU or when it has MetaData (and thus belongs to a generic connection).⌋()

Note: CAN FD and CAN frames will be mapped to different PDUs by CanIf depending on the frame format (CAN FD or CAN 2.0). Therefore, it is possible to distinguish between CAN FD and classic CAN communication.

7.3.8 N-PDU padding

To guarantee complete compatibility with all upper layer requirements concerning the frame data length (e.g. OBD requires data length to always be set to 8 bytes, however UDS does not), the padding activation is configurable at pre-compile time per N-SDU by using either CanTpRxPaddingActivation for a Rx N-SDU or CanTpTxPaddingActivation for a Tx N-SDU.

[SWS_CanTp_00116] ⌈In both padding and no padding modes, the CanTp module shall only transfer used data bytes to the upper layer.⌋(*SRS_Can_01073*)

[SWS_CanTp_00059] ⌈The value used for padding bytes is configurable via configuration parameter CANTP_PADDING_BYTE (see CanTpPaddingByte).⌋(*SRS_Can_01086*)

[SWS_CanTp_00344] ⌈If frames with a payload <= 8 (either CAN 2.0 frames or small CAN FD frames) are used for a Rx N-SDU and CanTpRxPaddingActivation is equal to CANTP_ON, then CanTp shall only accept SF Rx N-PDUs or last CF Rx N-PDUs, belonging to that N-SDU, with a length of eight bytes (i.e. PduInfoPtr.SduLength = 8).⌋(*SRS_Can_01073*)

[SWS_CanTp_00345] ⌈If frames with a payload <= 8 (either CAN 2.0 frames or small CAN FD frames) are used for a Rx N-SDU and CanTpRxPaddingActivation is equal to CANTP_ON, then CanTp receives by means of CanTp_RxIndication() call an SF Rx N-PDU belonging to that N-SDU, with a length smaller than eight bytes (i.e. PduInfoPtr.SduLength < 8), CanTp shall reject the reception. The runtime error code CanTp.CANTP_E_PADDING shall be reported to the Default Error Tracer.⌋(*SRS_Can_01073*)

[SWS_CanTp_00346] ⌈If frames with a payload <= 8 (either CAN 2.0 frames or small CAN FD frames) are used for a Rx N-SDU and CanTpRxPaddingActivation is

equal to CANTP_ON, and CanTp receives by means of CanTp_RxIndication() call a last CF Rx N-PDU belonging to that N-SDU, with a length smaller than eight bytes (i.e. PduInfoPtr. SduLength != 8), CanTp shall abort the ongoing reception by calling PduR_CanTpRxIndication() with the result E_NOT_OK. The runtime error code CanTp.CANTP_E_PADDING shall be reported to the Default Error Tracer.⌋*(SRS_Can_01073)*

[SWS_CanTp_00347] ⌈If CanTpRxPaddingActivation is equal to CANTP_ON for an Rx N-SDU, the CanTp module shall transmit FC N-PDUs with a length of eight bytes. Unused bytes in N-PDU shall be updated with CANTP_PADDING_BYTE (see CanTpPaddingByte).⌋*()*

[SWS_CanTp_00348] ⌈If frames with a payload <= 8 (either CAN 2.0 frames or small CAN FD frames) are used for a Tx N-SDU and if CanTpTxPaddingActivation is equal to CANTP_ON, CanTp shall transmit by means of CanIf_Transmit() call, SF Tx N-PDU or last CF Tx N-PDU that belongs to that Tx N-SDU with the length of eight bytes(i.e. PduInfoPtr.SduLength = 8). Unused bytes in N-PDU shall be updated with CANTP_PADDING_BYTE (see CanTpPaddingByte).⌋*(SRS_Can_01073)*

[SWS_CanTp_00349] ⌈If CanTpTxPaddingActivation is equal to CANTP_ON for a Tx N-SDU, and if a FC N-PDU is received for that Tx N-SDU on a ongoing transmission, by means of CanTp_RxIndication() call, and the length of this FC is smaller than eight bytes (i.e. PduInfoPtr.SduLength <8) the CanTp module shall abort the transmission session by calling PduR_CanTpTxConfirmation() with the result E_NOT_OK. The runtime error code CanTp.CANTP_E_PADDING shall be reported to the Default Error Tracer.⌋*()*

[SWS_CanTp_00351] ⌈If the data length which shall be transmitted via CanIf_Transmit() does not match possible DLC values (0..8, 12, 16, 20, 24, 32, 48, or 64), CanTp shall use the next higher valid DLC for transmission with initialization of unused bytes to the value of CANTP_PADDING_BYTE (see CanTpPaddingByte).⌋ *(SRS_Can_01073)*

Rationale: The ISO 11898-1:2015 DLC values from 9 to 15 are assigned to non-linear discrete values for CAN frame payload length up to 64 byte. To prevent the transmission of uninitialized data the padding of CAN frame data is mandatory for DLC values greater than eight when the length of the N_PDU size to be transmitted is not equal to one of the discrete length values defined in the ISO 11898-1:2015 DLC table. For DLC values from 9 to 15 only the mandatory padding should be used.

The following picture represents an ISO frame:

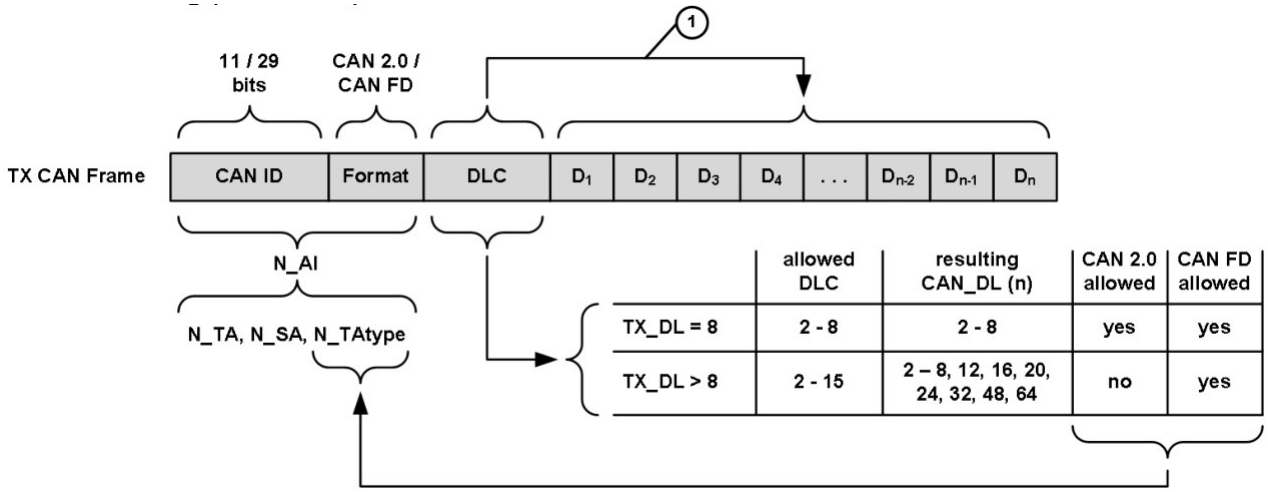

Figure 7.9: ISO frame construction

7.3.9 Handling of unexpected N-PDU arrival

The behavior of the CAN Transport Layer on unexpected N-PDU arrival is greatly dependent on the communication direction type of the processing N-SDU.

[SWS_CanTp_00057] ⌈If unexpected frames are received, the CanTp module shall behave according to the table below. This table specifies the N-PDU handling considering the current CanTp internal status (CanTp status).⌋*(SRS_Can_01082)*

It must be understood, that the received N-PDU contains the same address information (N_AI) as the reception or transmission, which may be in progress at the time the N_PDU is received.

CanTp	Reception of				
status	**SF N-PDU**	**FF N-PDU**	**CF N-PDU**	**FC N-PDU**	**Unknown N-PDU**
Segmented Transmit in progress	If a reception is in progress process it according to the cell below, otherwise process the SF N-PDU as the start of a new reception	If a reception is in progress process it according to the cell below, otherwise process the FF N-PDU as the start of a new reception	If a reception is in progress process it according to the cell below, otherwise ignore it.	If awaited, process the FC N-PDU, otherwise ignore it.	Ignore

▽

△

CanTp	Reception of				
Segmented Receive in progress	Terminate the current reception, report an indication, with parameter Result set to E_NOT_OK, to the upper layer, and process the SF N-PDU as the start of a new reception	Terminate the current reception, report an indication, with parameter Result set to E_NOT_OK, to the upper layer, and process the FF N-PDU as the start of a new reception	Process the CF N-PDU in the on-going reception and perform the required checks (e.g. SN in right order)	If a transmission is in progress process it according to the cell above, otherwise ignore it.	Ignore
$Idle$[1]	Process the SF N-PDU as the start of a new reception	Process the FF N-PDU as the start of a new reception	Ignore	Ignore	Ignore

[1]

Table 1: Handling of N-PDU arrivals

7.4 Error Classification

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

[SWS_CanTp_00008] ⌈On errors and exceptions, the CanTp module shall not modify its current module state (see Figure 3: CAN Transport Layer life cycle) but shall simply report the error event.⌋*(SRS_BSW_00339)*

[SWS_CanTp_00291] ⌈In case of production error, the Diagnostic Event Manager module (via the Function Inhibition Manager) will perform the appropriate action (e.g. status modification of the calling module).⌋*()*

7.4.1 Development Errors

[SWS_CanTp_00293] ⌈

Type of error	*Related error code*	*Error value*
API service called with wrong parameter(s): When CanTp_ChangeParameter is called with invalid value.	CANTP_E_PARAM_CONFIG	0x01
API service called with wrong parameter(s): When CanTp_ChangeParameter or CanTp_Read Parameter is called with invalid parameter ID.	CANTP_E_PARAM_ID	0x02

▽

[1] Idle = CANTP_ON.CANTP_RX_WAIT and CANTP_ON.CANTP_TX_WAIT

△

Type of error	*Related error code*	*Error value*
API service called with a NULL pointer.	CANTP_E_PARAM_POINTER	0x03
Module initialization has failed, e.g. CanTp_Init() called with an invalid pointer in postbuild.	CANTP_E_INIT_FAILED	0x04
API service used without module initialization : On any API call except CanTp_Init(), CanTp_Get VersionInfo() and CanTp_MainFunction() if CanTp is in state CANTP_OFF	CANTP_E_UNINIT	0x20
Invalid Transmit PDU identifier (e.g. a service is called with an inexistent Tx PDU identifier)	CANTP_E_INVALID_TX_ID	0x30
Invalid Receive PDU identifier (e.g. a service is called with an inexistent Rx PDU identifier)	CANTP_E_INVALID_RX_ID	0x40

⌋()

7.4.2 Runtime Errors

[SWS_CanTp_00352] ⌈

Type of error	*Related error code*	*Error value*
PDU received with a length smaller than 8 bytes. (i.e. PduInfoPtr.SduLength < 8)	CANTP_E_PADDING	0x70
CanTp_Transmit() is called for a configured Tx I-Pdu with functional addressing and the length parameter indicates, that the message can not be sent with a SF	CANTP_E_INVALID_TATYPE	0x90
Requested operation is not supported - a cancel transmission/reception request for an N-SDU that it is not on transmission/reception process	CANTP_E_OPER_NOT_SUPPORTED	0xA0
Event reported in case of an implementation specific error other than a protocol timeout error during a reception or a transmission	CANTP_E_COM	0xB0
Event reported in case of a protocol timeout error during reception	CANTP_E_RX_COM	0xC0
Event reported in case of a protocol timeout error during transmission	CANTP_E_TX_COM	0xD0

⌋()

[SWS_CanTp_00229] ⌈If the task was aborted due to As, Bs, Cs, Ar, Br, Cr timeout, the CanTp module shall raise the development error `CanTp.CANTP_E_RX_COM` (in case of a reception operation) or `CanTp.CANTP_E_TX_COM` (in case of a transmission operation). If the task was aborted due to any other protocol error, the CanTp module shall raise the runtime error code `CanTp.CANTP_E_COM` to the Default Error Tracer.⌋
()

7.4.3 Transient Faults

There are no transient faults.

7.4.4 Production Errors

There are no production errors.

7.4.5 Extended Production Errors

There are no extended production errors.

8 API specification

8.1 Imported types

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

[SWS_CanTp_00209] ⌈

Module	*Header File*	*Imported Type*
ComStack_Types	ComStack_Types.h	BufReq_ReturnType
	ComStack_Types.h	PduIdType
	ComStack_Types.h	PduInfoType
	ComStack_Types.h	PduLengthType
	ComStack_Types.h	RetryInfoType
	ComStack_Types.h	TPParameterType
	ComStack_Types.h	TpDataStateType
Std	Std_Types.h	Std_ReturnType
	Std_Types.h	Std_VersionInfoType

⌋() In order to receive a consistent API for the AUTOSAR communication stack, basic types have been defined. These types are used by the CAN Transport Layer to communicate with the Pdu-Router and with the CAN Interface Layer.

For more information, these basic types are presented in depth in the AUTOSAR COM stack API specification.

These AUTOSAR standard types will be used without any type redefinition.

[SWS_CanTp_00002] ⌈If, for implementation reasons, some additional types have to be defined, the CanTp module shall label these types as follows: `CanTp_<TypeName>Type`, where `<TypeName>` is the name of this type adhering to the rules:

- No underscore usage
- First letter of each word upper case, consecutive letters lower case.

⌋(*SRS_BSW_00353*)

[SWS_CanTp_00296] ⌈The CanTp module shall ensure that implementation-specific types are not "visible" outside of CanTp. Otherwise, the complete architecture would be corrupted.⌋*()*

8.2 Type definitions

8.2.1 CanTp_ConfigType

[SWS_CanTp_00340] ⌈

Name	CanTp_ConfigType
Kind	Structure
Description	Data structure type for the post-build configuration parameters.
Available via	CanTp.h

⌋*()* Implementation specific data structure type for the post-build configuration parameters.

8.3 Function definitions

This is a list of functions provided for upper layer modules

8.3.1 CanTp_Init

[SWS_CanTp_00208] ⌈

Service Name	CanTp_Init	
Syntax	`void CanTp_Init ( const CanTp_ConfigType* CfgPtr )`	
Service ID [hex]	0x01	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	CfgPtr	Pointer to the CanTp post-build configuration data.
Parameters (inout)	None	
Parameters (out)	None	

▽

△

Return value	None
Description	This function initializes the CanTp module.
Available via	CanTp.h

⌋*(SRS_BSW_00101, SRS_BSW_00358, SRS_BSW_00414)* After power up, CanTp is in a state called `CANTP_OFF` (see [SWS_CanTp_00168]). In this state, the CanTp is not yet configured and therefore cannot perform any communication task.

The function `CanTp_Init` initializes all global variables of the CAN Transport Layer with the given configuration set and set it in the idle state (state = `CANTP_ON` but neither transmission nor reception are in progress) (see [SWS_CanTp_00170] and [SWS_CanTp_00030]).

The function `CanTp_Init` has no return value because configuration data errors should be detected during configuration time (e.g. by the configuration tools). Furthermore, if a hardware error occurs, it will be reported via the error manager modules.

[SWS_CanTp_00199] ⌈The CanTp module's environment shall call `CanTp_Init` before using the CanTp module for further processing.⌋*()*

Parameter checking for the initialization function is specified within BSW General [4].

[SWS_CanTp_00161] ⌈A static status variable, denoting whether a BSW module is initialized, should be initialized with value 0 before any APIs of the BSW module are called.

The initialization function of the BSW modules will set the static status variable to a value not equal to 0.

This variable is used to check if the module has been initialized before calling an API.⌋ *(SRS_BSW_00406)*

8.3.2 CanTp_ GetVersionInfo

[SWS_CanTp_00210] ⌈

Service Name	CanTp_GetVersionInfo

Syntax	`void CanTp_GetVersionInfo (`
	`  Std_VersionInfoType* versioninfo`
	`)`

Service ID [hex]	0x07
Sync/Async	Synchronous
Reentrancy	Reentrant
Parameters (in)	None
Parameters (inout)	None

▽

△

Parameters (out)	versioninfo	Indicator as to where to store the version information of this module.
Return value	None	
Description	This function returns the version information of the CanTp module.	
Available via	CanTp.h	

⌋()

[SWS_CanTp_00319] ⌈If development error detection is enabled the function `CanTp_GetVersionInfo` shall raise `CanTp.CANTP_E_PARAM_POINTER` error if the argument is a NULL pointer.⌋()

8.3.3 CanTp_Shutdown

[SWS_CanTp_00211] ⌈

Service Name	CanTp_Shutdown
Syntax	`void CanTp_Shutdown ( void )`
Service ID [hex]	0x02
Sync/Async	Synchronous
Reentrancy	Non Reentrant
Parameters (in)	None
Parameters (inout)	None
Parameters (out)	None
Return value	None
Description	This function is called to shutdown the CanTp module.
Available via	CanTp.h

⌋()

[SWS_CanTp_00202] ⌈The function `CanTp_Shutdown` shall close all pending transport protocol connections, free all resources and set the CanTp module into the `CANTP_OFF` state.⌋()

[SWS_CanTp_00200] ⌈The function `CanTp_Shutdown` shall not raise a notification about the pending frame transmission or reception.⌋()

8.3.4 CanTp_Transmit

[SWS_CanTp_00212] ⌈

Service Name	CanTp_Transmit	
Syntax	`Std_ReturnType CanTp_Transmit ( PduIdType TxPduId, const PduInfoType* PduInfoPtr )`	
Service ID [hex]	0x49	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different PduIds. Non reentrant for the same PduId.	
Parameters (in)	TxPduId	Identifier of the PDU to be transmitted
	PduInfoPtr	Length of and pointer to the PDU data and pointer to MetaData.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: Transmit request has been accepted. E_NOT_OK: Transmit request has not been accepted.
Description	Requests transmission of a PDU.	
Available via	CanTp.h	

⌋()

[SWS_CanTp_00231] ⌈If data does fit into the associated N-PDU, the function CanTp_Transmit() shall send a SF N-PDU.⌋(*SRS_Can_01071, SRS_Can_01074*)

[SWS_CanTp_00232] ⌈If data does not fit into the associated N-PDU, the function CanTp_Transmit() shall initiate a multiple frame transmission session.⌋(*SRS_Can_01071, SRS_Can_01074*)

[SWS_CanTp_00354] ⌈The maximum Tx length of the N-PDU shall be derived from the PduLength configuration parameter of EcuC. This parameter is equivalent to TX_DL of ISO 15765-2.⌋(*SRS_Can_01163*)

[SWS_CanTp_00204] ⌈The CanTp module shall notify the upper layer by calling the PduR_CanTpTxConfirmation callback when the transmit request has been completed.⌋()

[SWS_CanTp_00205] ⌈The CanTp module shall abort the transmit request and call the PduR_CanTpTxConfirmation callback function with the appropriate error result value if an error occurred (over flow, N_As timeout, N_Bs timeout and so on).⌋()

The error result values are defined according to the ISO 15765 and definition of this type is depicted in the ComStackTypes document (AUTOSAR_SWS_ComStackTypes, Chapter 8.1.5).

[SWS_CanTp_00206] ⌈The function CanTp_Transmit shall reject a request if the CanTp_Transmit service is called for a N-SDU identifier which is being used in a currently running CAN Transport Layer session.⌋()

[SWS_CanTp_00298] ⌈CanTp has limited buffering capability, and hence the N-SDU payload to be transmitted is not copied internally. The CAN Transport Layer obtains the data directly from the upper layer via the PduR_CanTpCopyTxData service.⌋()

Thus, to guarantee the data consistency, the upper layer (e.g. DCM, PduRouter or AUTOSAR COM) must lock this memory area until the confirmation notification occurs.

[SWS_CanTp_00299] ⌈When the upper layer calls this function for an N-SDU without `MetaData`, only the data length information of the structure indicated by `PduInfoPtr` has to be used. Its value indicates the payload length of the N-SDU, which is to be transmitted.

For an N-SDU with `MetaData`, besides the length information also the MetaData provided via the PduInfoPtr is relevant. To obtain actual Tx data, the CAN Transport Layer shall call the `PduR_CanTpCopyTxData` service.⌋*()*

[SWS_CanTp_00321] ⌈If development error detection is enabled the function `CanTp_Transmit` shall raise `CanTp.CANTP_E_PARAM_POINTER` error if the argument `PduInfoPtr` is a NULL pointer.⌋*()*

[SWS_CanTp_00356] ⌈If development error detection is enabled the function `CanTp_Transmit` shall check the validity of function parameter `TxPduId`. If its value is invalid, the `CanTp_Transmit` function shall raise the development error `CanTp.CANTP_E_INVALID_TX_ID`.⌋*()*

8.3.5 CanTp_CancelTransmit

[SWS_CanTp_00246] ⌈

Service Name	CanTp_CancelTransmit	
Syntax	`Std_ReturnType CanTp_CancelTransmit ( PduIdType TxPduId )`	
Service ID [hex]	0x4a	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different PduIds. Non reentrant for the same PduId.	
Parameters (in)	TxPduId	Identification of the PDU to be cancelled.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: Cancellation was executed successfully by the destination module. E_NOT_OK: Cancellation was rejected by the destination module.
Description	Requests cancellation of an ongoing transmission of a PDU in a lower layer communication module.	
Available via	CanTp.h	

⌋*()* This service cancels the transmission of an N-SDU that has already requested for transmission by calling `CanTp_Transmit` service.

[SWS_CanTp_00254] ⌈If development error detection is enabled the function `CanTp_CancelTransmit` shall check the validity of function parameter `TxPduId`. If

its value is invalid, the CanTp_CancelTransmit function shall raise the development error CanTp.CANTP_E_INVALID_TX_ID.

If the parameter value indicates a cancel transmission request for an N-SDU that it is not on transmission process the CanTp module shall report a runtime error code CanTp.CANTP_E_OPER_NOT_SUPPORTED to the Default Error Tracer and the service shall return E_NOT_OK.⌋()

[SWS_CanTp_00256] ⌈The CanTp shall abort the transmission of the current N-SDU if the service returns E_OK.⌋()

[SWS_CanTp_00255] ⌈If the CanTp_CancelTransmit service has been successfully executed the CanTp shall call the PduR_CanTpTxConfirmation with notification result E_NOT_OK.⌋()

8.3.6 CanTp_CancelReceive

[SWS_CanTp_00257] ⌈

Service Name	CanTp_CancelReceive	
Syntax	`Std_ReturnType CanTp_CancelReceive ( PduIdType RxPduId )`	
Service ID [hex]	0x4c	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	RxPduId	Identification of the PDU to be cancelled.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: Cancellation was executed successfully by the destination module. E_NOT_OK: Cancellation was rejected by the destination module.
Description	Requests cancellation of an ongoing reception of a PDU in a lower layer transport protocol module.	
Available via	CanTp.h	

⌋() The service CanTp_CancelReceive cancels the reception of an N-SDU initiated by the reception of a First Frame and consequently calls of PduR_StartOfReception. When the function returns, no reception is in progress anymore with the given N-SDU identifier.

[SWS_CanTp_00260] ⌈If development error detection is enabled the function CanTp_CancelReceive shall check the validity of function parameter RxPduId. If its value is invalid, the CanTp_CancelReceive function shall raise the development error CanTp.CANTP_E_INVALID_RX_ID.

If the parameter value indicates a cancel reception request for an N-SDU that it is not on reception process the CanTp module shall report the runtime error code CanTp_CANTP_E_OPER_NOT_SUPPORTED to the Default Error Tracer and the service shall return E_NOT_OK.⌋()

[SWS_CanTp_00261] ⌈The CanTp shall abort the reception of the current N-SDU if the service returns E_OK.⌋()

[SWS_CanTp_00262] ⌈The CanTp shall reject the request for receive cancellation in case of a Single Frame reception or if the CanTp is in the process of receiving the last Consecutive Frame of the N-SDU (i.e. the service is called after N-Cr timeout is started for the last Consecutive Frame). In this case the CanTp shall return E_NOT_OK.⌋()

[SWS_CanTp_00263] ⌈If the CanTp_CancelReceive service has been successfully executed the CanTp shall call the PduR_CanTpRxIndication with notification result E_NOT_OK.⌋()

8.3.7 CanTp_ChangeParameter

[SWS_CanTp_00302] ⌈

Service Name	CanTp_ChangeParameter	
Syntax	`Std_ReturnType CanTp_ChangeParameter ( PduIdType id, TPParameterType parameter, uint16 value )`	
Service ID [hex]	0x4b	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	id	Identification of the PDU which the parameter change shall affect.
	parameter	ID of the parameter that shall be changed.
	value	The new value of the parameter.
Parameters (inout)	None	
Parameters (out)	None	
Return value	Std_ReturnType	E_OK: The parameter was changed successfully. E_NOT_OK: The parameter change was rejected.
Description	Request to change a specific transport protocol parameter (e.g. block size).	
Available via	CanTp.h	

⌋() The service CanTp_ChangeParameter is used to change the value of the reception parameter BS and STmin associated to each received N-SDU.

Implementation of this service depends on the configuration parameter CanTpChangeParameterApi (i.e. the service shall be implemented when the parameter is set to TRUE).

[SWS_CanTp_00303] ⌈A parameter change is only possible if the related N-SDU is not in the process of reception - i.e. a change of parameter value it is not possible after reception of FF until indication for last CF reception of the related N-SDU.⌋*()*

[SWS_CanTp_00304] ⌈If the change of a parameter is requested for an N-SDU that is on reception process the service `CanTp_ChangeParameter` immediately returns `E_NOT_OK` and no parameter value is changed⌋*()*

[SWS_CanTp_00338] ⌈When `CanTp_ChangeParameter` is called for an N-SDU with `MetaData` (indicating a generic connection), the change shall be applied to all generic connections, so that it is used for all following receptions.⌋*()*

[SWS_CanTp_00305] ⌈If development error detection is enabled, the function `CanTp_ChangeParameter` shall check the validity of function parameters (`parameter` and `value`). If the `parameter` parameter is invalid, the `CanTp_ChangeParameter` function shall raise the development error `CanTp.CANTP_E_PARAM_ID`. If the `value` parameter is invalid, the `CanTp_ChangeParameter` function shall raise the development error `CanTp.CANTP_E_PARAM_CONFIG`.⌋*()*

[SWS_CanTp_00357] ⌈If development error detection is enabled the function `CanTp_ChangeParameter` shall check the validity of function parameter `id`. If its value is invalid, the `CanTp_ChangeParameter` function shall raise the development error `CanTp.CANTP_E_INVALID_RX_ID`.⌋*()*

8.3.8 CanTp_ReadParameter

[SWS_CanTp_00323] ⌈

Service Name	CanTp_ReadParameter	
Syntax	`Std_ReturnType CanTp_ReadParameter ( PduIdType id, TPParameterType parameter, uint16* value )`	
Service ID [hex]	0x0b	
Sync/Async	Synchronous	
Reentrancy	Non Reentrant	
Parameters (in)	id	Identifier of the received N-SDU on which the reception parameter are read.
	parameter	Specify the parameter to which the value has to be read (BS or STmin).
Parameters (inout)	None	
Parameters (out)	value	Pointer where the parameter value will be provided.
Return value	Std_ReturnType	E_OK: request is accepted. E_NOT_OK: request is not accepted.
Description	This service is used to read the current value of reception parameters BS and STmin for a specified N-SDU.	
Available via	CanTp.h	

⌋() Implementation of this service depends on the configuration parameter CanTpChangeParameterApi (i.e. the service shall be implemented when the parameter is set to TRUE).

[SWS_CanTp_00324] ⌈If development error detection is enabled the function CanTp_ReadParameter shall check the validity of function parameter parameter. If its value is invalid, the CanTp_ReadParameter function shall raise the development error CanTp.CANTP_E_PARAM_ID.⌋()

[SWS_CanTp_00358] ⌈If development error detection is enabled the function CanTp_ReadParameter shall check the validity of function parameter id. If its value is invalid, the CanTp_ReadParameter function shall raise the development error CanTp.CANTP_E_INVALID_RX_ID.⌋()

8.3.9 Main Function

[SWS_CanTp_00213] ⌈

Service Name	CanTp_MainFunction
Syntax	`void CanTp_MainFunction ( void )`
Service ID [hex]	0x06
Description	The main function for scheduling the CAN TP.
Available via	SchM_CanTp.h

⌋()

[SWS_CanTp_00164] ⌈The main function for scheduling the CAN TP (Entry point for scheduling)

The main function will be called by the Schedule Manager or by the Free Running Timer module according of the call period needed. CanTp_MainFunction is involved in handling of CAN TP timeouts N_As, N_Bs, N_Cs, N_Ar, N_Br,

N_Cr and STMmin.⌋(*SRS_BSW_00424, SRS_BSW_00373*)

[SWS_CanTp_00300] ⌈The function CanTp_MainFunction is affected by configuration parameter CanTpMainFunctionPeriod.⌋()

8.4 Callback notifications

The following is a list of functions provided for lower layer modules.

8.4.1 CanTp_RxIndication

[SWS_CanTp_00214] ⌈

Service Name	CanTp_RxIndication	
Syntax	`void CanTp_RxIndication ( PduIdType RxPduId, const PduInfoType* PduInfoPtr )`	
Service ID [hex]	0x42	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different PduIds. Non reentrant for the same PduId.	
Parameters (in)	RxPduId	ID of the received PDU.
	PduInfoPtr	Contains the length (SduLength) of the received PDU, a pointer to a buffer (SduDataPtr) containing the PDU, and the MetaData related to this PDU.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	Indication of a received PDU from a lower layer communication interface module.	
Available via	CanTp.h	

⌋*()* The CanIf module shall call this function after a successful reception of a Rx CAN L-PDU.

The data will be copied by the CanTp via the PDU structure `PduInfoType`. In this case the L-PDU buffers are not global and are therefore distributed in the corresponding CAN Transport Layer.

Note that `PduInfoPtr` contains also the `MetaData` in case of dynamic Rx N-PDUs.

[SWS_CanTp_00235] ⌈The function `CanTp_RxIndication` shall be callable in interrupt context (it could be called from the CAN receive interrupt).⌋*()*

[SWS_CanTp_00322] ⌈If development error detection is enabled the function `CanTp_RxIndication` shall raise `CanTp.CANTP_E_PARAM_POINTER` error if the argument `PduInfoPtr` is a NULL pointer.⌋*()*

[SWS_CanTp_00359] ⌈If development error detection is enabled the function `CanTp_RxIndication` shall check the validity of function parameter `RxPduId`. If its value is invalid, the `CanTp_RxIndication` function shall raise the development error `CanTp.CANTP_E_INVALID_RX_ID`.⌋*()*

8.4.2 CanTp_TxConfirmation

[SWS_CanTp_00215] ⌈

Service Name	CanTp_TxConfirmation	
Syntax	`void CanTp_TxConfirmation ( PduIdType TxPduId, Std_ReturnType result )`	
Service ID [hex]	0x40	
Sync/Async	Synchronous	
Reentrancy	Reentrant for different PduIds. Non reentrant for the same PduId.	
Parameters (in)	TxPduId	ID of the PDU that has been transmitted.
	result	E_OK: The PDU was transmitted. E_NOT_OK: Transmission of the PDU failed.
Parameters (inout)	None	
Parameters (out)	None	
Return value	None	
Description	The lower layer communication interface module confirms the transmission of a PDU, or the failure to transmit a PDU.	
Available via	CanTp.h	

⌋() The CanIf module shall call the function `CanTp_TxConfirmation` after the TP related CAN Frame (SF, FF, CF, FC) has been transmitted through the CAN network.

[SWS_CanTp_00236] ⌈The function `CanTp_TxConfirmation` shall be callable in interrupt context (it could be called from the CAN transmit interrupt).⌋()

[SWS_CanTp_00360] ⌈If development error detection is enabled the function `CanTp_TxConfirmation` shall check the validity of function parameter `TxPduId`. If its value is invalid, the `CanTp_TxConfirmation` function shall raise the development error `CanTp.CANTP_E_INVALID_TX_ID`.⌋()

8.5 Expected interfaces

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

8.5.1 Mandatory Interfaces

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

[SWS_CanTp_00216] ⌈

API Function	*Header File*	*Description*
CanIf_Transmit	CanIf.h	Requests transmission of a PDU.
Det_ReportRuntimeError	Det.h	Service to report runtime errors. If a callout has been configured then this callout shall be called.

▽

△

API Function	Header File	Description
PduR_CanTpCopyRxData	PduR_CanTp.h	This function is called to provide the received data of an I-PDU segment (N-PDU) to the upper layer. Each call to this function provides the next part of the I-PDU data. The size of the remaining buffer is written to the position indicated by bufferSizePtr.
PduR_CanTpCopyTxData	PduR_CanTp.h	This function is called to acquire the transmit data of an I-PDU segment (N-PDU). Each call to this function provides the next part of the I-PDU data unless retry->TpDataState is TP_DATARETRY. In this case the function restarts to copy the data beginning at the offset from the current position indicated by retry->TxTpDataCnt. The size of the remaining data is written to the position indicated by availableDataPtr.
PduR_CanTpRxIndication	PduR_CanTp.h	Called after an I-PDU has been received via the TP API, the result indicates whether the transmission was successful or not.
PduR_CanTpStartOfReception	PduR_CanTp.h	This function is called at the start of receiving an N-SDU. The N-SDU might be fragmented into multiple N-PDUs (FF with one or more following CFs) or might consist of a single N-PDU (SF). The service shall provide the currently available maximum buffer size when invoked with TpSdu Length equal to 0.
PduR_CanTpTxConfirmation	PduR_CanTp.h	This function is called after the I-PDU has been transmitted on its network, the result indicates whether the transmission was successful or not.

⌋()

8.5.2 Optional Interfaces

This chapter defines the interface, which is required, in order to fulfill the optional functionality of the module.

[SWS_CanTp_00217] ⌈

API Function	Header File	Description
Det_ReportError	Det.h	Service to report development errors.

⌋()

9 Sequence diagrams

The goal of this chapter is to make it easier to understand the CAN Transport Layer by describing most of the more frequent and complicated use cases. Thus, the following diagram sequences are not exhaustive and do not reflect all the specified API possibilities.

9.1 SF N-SDU received and no buffer available.

9.1.1 Assumptions

- All input parameters are OK;
- The N-SDU data length fits into the associated N-PDU;
- Upper layer can not make an Rx buffer available.

9.1.2 Sequence diagram

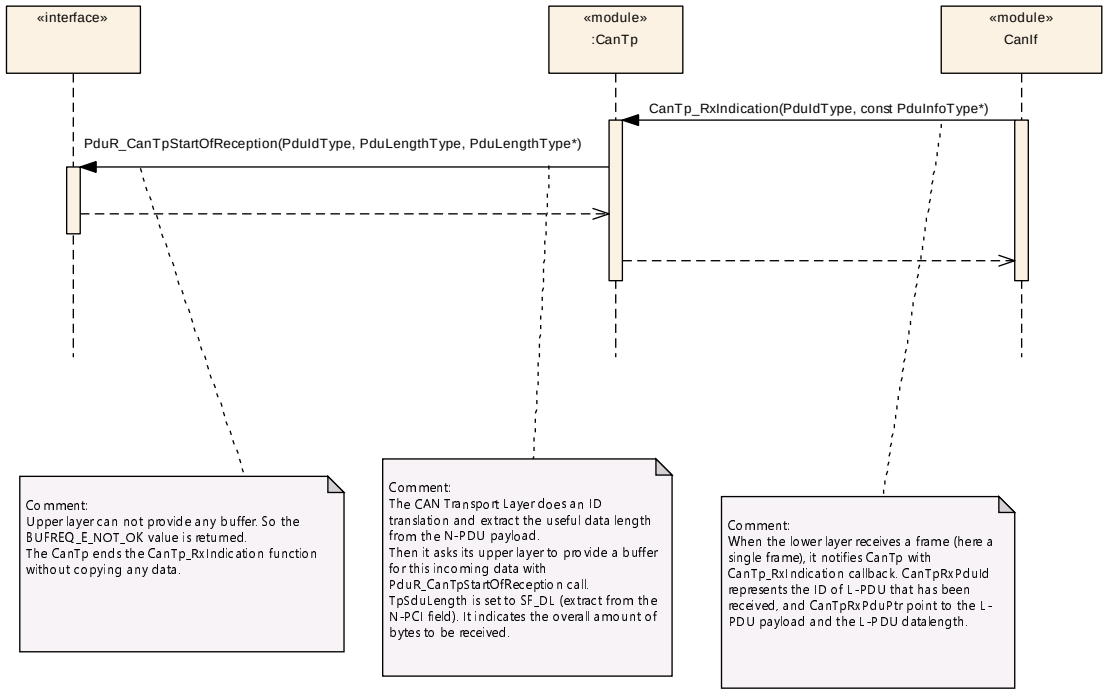

Figure 9.1: SF N-SDU received and no buffer available

Note: This sequence diagram demonstrates the working of the CAN_Tp module only. However, if the whole system is considered during such reception, more modules are involved. Since this reception can be triggered in the context of CAN ISR, the CAN_Tp operation should be as short as possible.

9.1.3 Transition description

Transition	Name	Description
1	CanTp_RxIndication (RxPduId, PduInfoPtr)	When the lower layer receives a frame (here a single frame), it notifies CanTp by means of a CanTp_RxIndication callback. RxPduId represents the ID of L-PDU that has been received, and PduInfoPtr indicates the L-PDU payload and the L-PDU data length.
2	PduR_CanTpStartOfReception(id, info, TpSduLength, bufferSizePtr)	The CAN Transport Layer performs an ID translation and extracts the useful data length from the N-PDU payload. It then asks its upper layer to make a buffer available for this incoming data with a PduR_CanTpStartOfReception callback. TpSduLength is set to SF_DL (extracted from the N-PCI field). It indicates the overall amount of bytes to be received.
3	BUFREQ_E_NOT_OK	The upper layer cannot make any buffer available, so the BUFREQ_E_ NOT_OK value is returned. The CanTp ends the CanTp_Rx Indication function without copying any data.

9.2 Successful SF N-PDU reception

9.2.1 Assumptions

- All input parameters are OK;
- The N-SDU data length fits into the associated N-PDU;
- The SF N-PDU is successfully received.

9.2.2 Sequence diagram

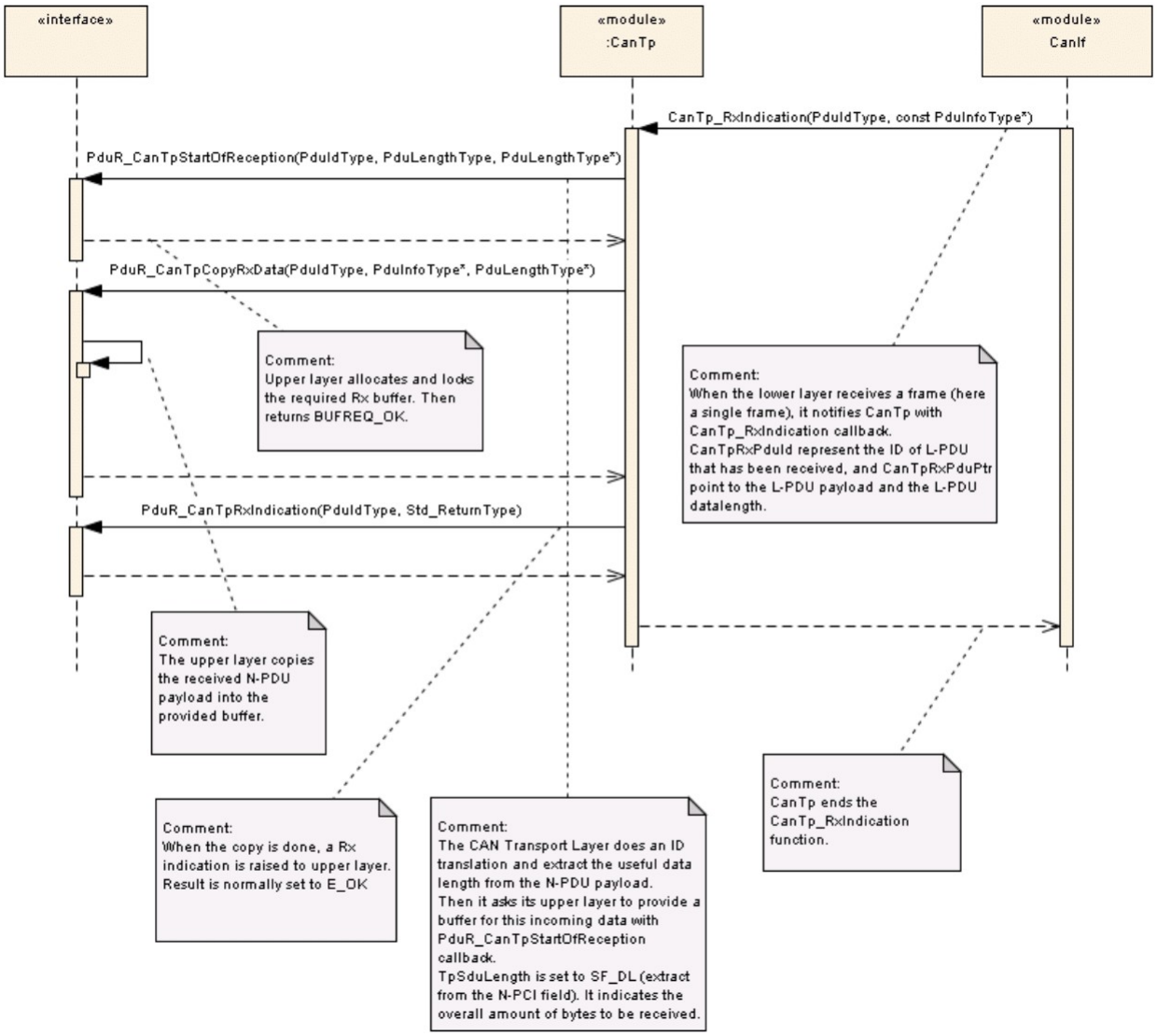

Figure 9.2: Successful SF N-PDU reception

Note: This sequence diagram demonstrates the working of the CAN_Tp module only. However, if the whole system is considered during such reception, more modules are involved. Since this reception can be triggered in the context of CAN ISR, the CAN_Tp operation should be as short as possible.

9.2.3 Transition description

Transition	Name	Description
1	CanTp_RxIndication (RxPduId, PduInfoPtr)	When the lower layer receives a frame (here a single frame), it notifies CanTp by means of a CanTp_RxIndication callback. RxPduId represents the ID of the L-PDU that has been received, and PduInfoPtr indicates the L-PDU payload and the L-PDU data length.
2	PduR_CanTpStartOfReception(id, info, TpSduLength, bufferSizePtr)	The CAN Transport Layer performs an ID translation and extracts the useful data length from the N-PDU payload. It then asks its upper layer to make a buffer available for this incoming data with a PduR_CanTpStartOfReception callback. TpSduLength is set to SF_DL (extracted from the N-PCI field). It indicates the overall amount of bytes to be received.
3	BUFREQ_OK	Upper layer allocates and locks the required Rx buffer. Then returns BUFREQ_OK.
4	PduR_CanTpCopyRxData(id, info, bufferSizePtr)	The upper layer copies the received N-PDU payload into the buffer.
5	PduR_CanTpRxIndication (id, result)	When the copy is complete, an Rx indication is sent to the upper layer. The result is set to E_OK.
6		CanTp ends the CanTp_RxIndication function.

9.3 Transmit request of SF N-SDU

9.3.1 Assumptions

- All input parameters are OK;
- The N-SDU data length fits into the associated N-PDU;
- The transmission is successfully processed.

9.3.2 Sequence diagram

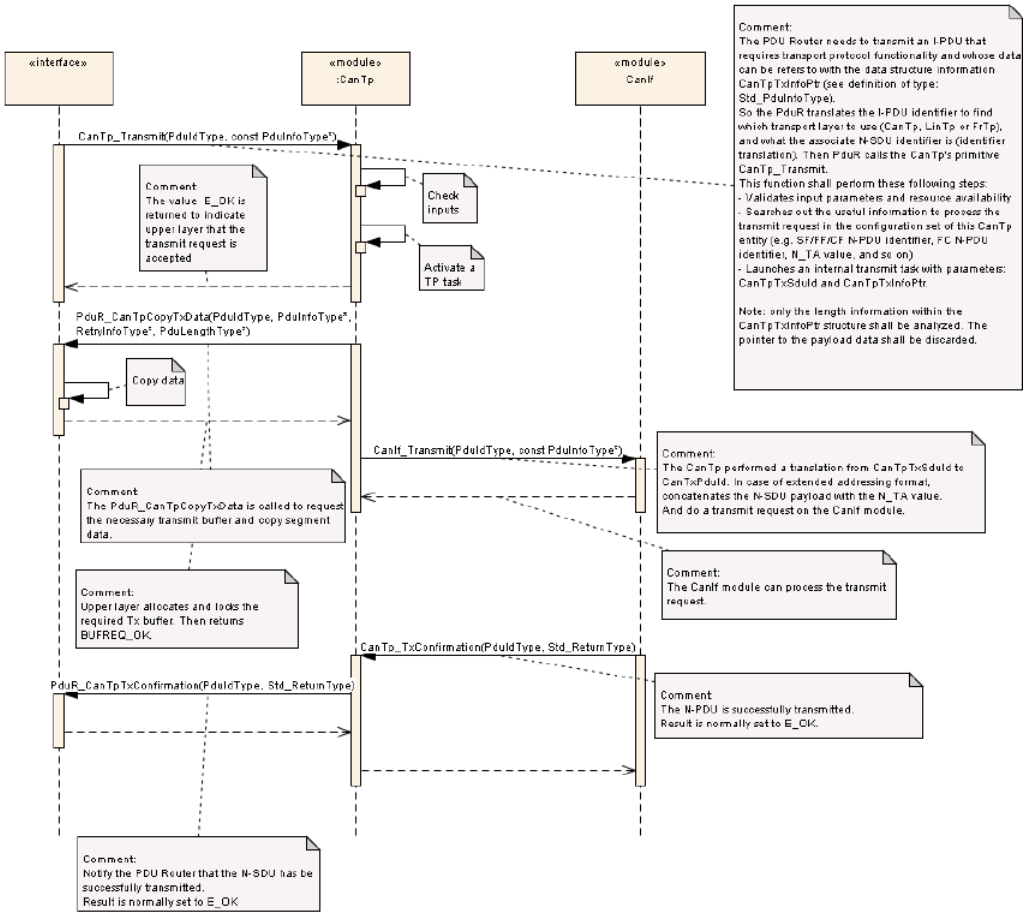

Figure 9.3: Transmit request of SF N-SDU

9.3.3 Transition description

Transition	Name	Description
1	CanTp_Transmit(TxPduId, PduInfoPtr)	The PDU Router needs to transmit an I-PDU that requires transport protocol functionality and whose data can be refers to with the data structure information PduInfoPtr (see definition of type: Std_PduInfoType). So the PduR translates the I-PDU identifier to find which transport layer to use (CanTp, LinTp or FrTp), and what the associate N-SDU identifier is (identifier translation). Then PduR calls the CanTp's primitive CanTp_Transmit.

△

Transition	Name	Description
		△ This function shall perform these following steps: - Validates input parameters and resource availability - Searches out the useful information to process the transmit request in the configuration set of this CanTp entity (e.g. SF/FF/CF N-PDU identifier, FC N-PDU identifier, N_TA value, and so on) - Launches an internal transmit task with parameters: TxPduId and PduInfo Ptr. Note: only the length information within the PduInfoPtr structure shall be analyzed. The pointer to the payload data shall be discarded.
2	E_OK	The value E_OK is returned to indicate to the upper layer that the transmit request is accepted. The upper layer locks the required Tx buffer.
3	PduR_CanTpCopyTxdata (id, info, retry, availableDataPtr)	The PduR_CanTpCopyTxData is called to copy segment data.
4	BUFREQ_OK	Upper layer copy data, then returns BUFREQ_OK.
5	CanIf_Transmit(TxPduId, PduInfoPtr)	The CanTp performs a translation of the TxPduId. In case of extended addressing format, it concatenates the N-SDU payload with the N_TA value, to perform a transmit request on the Can If module.
6	E_OK	The CanIf module can process the transmit request.
7	CanTp_TxConfirmation(TxPduId, result)	The N-PDU is successfully transmitted.
8	PduR_CanTpTxConfirmation (id, result)	Notifies the PDU Router that the N-SDU has been successfully transmitted. Consequently, the PduInfo Type structure has to be unlocked. Result is set to E_OK.

9.4 Transmit request of larger N-SDU

9.4.1 Assumptions

- All input parameters are OK;
- The N-SDU data length does not fit into the associated N-PDU;
- The transmission is successfully processed.

9.4.2 Sequence diagram

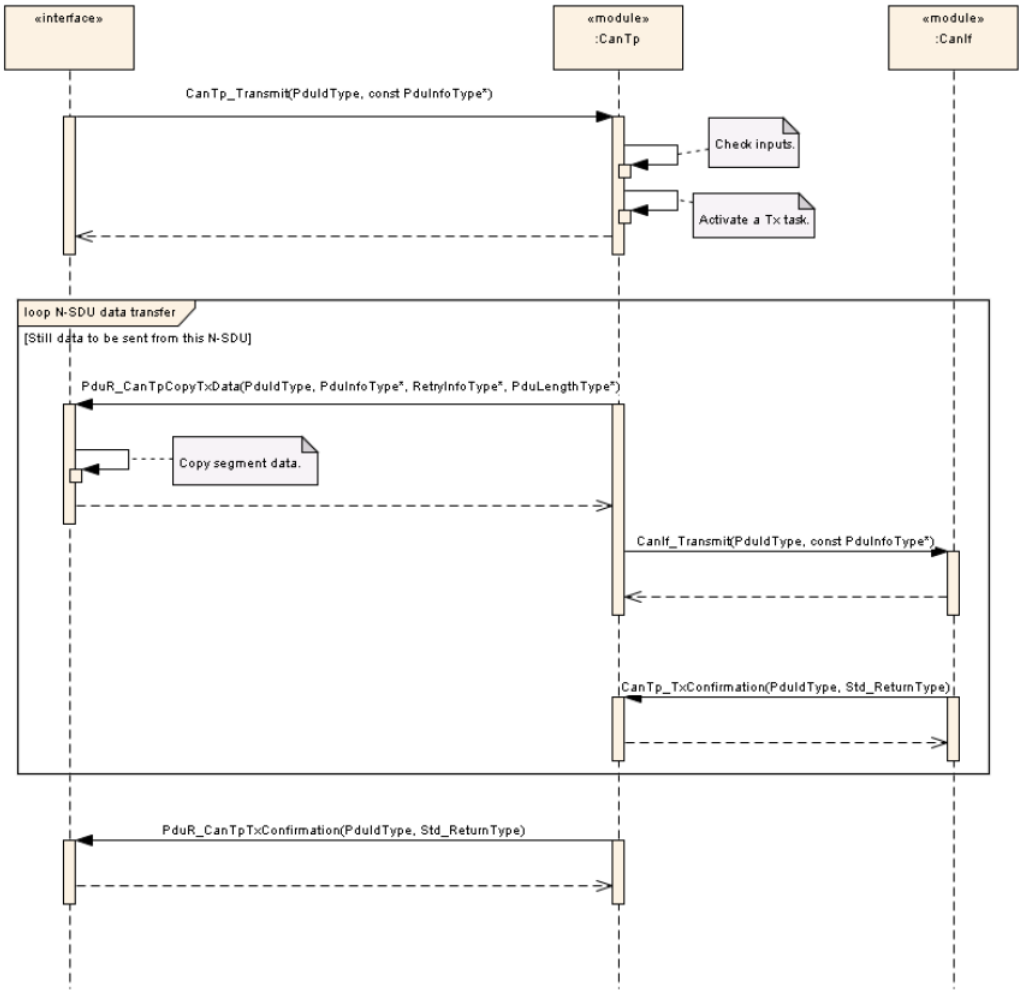

Figure 9.4: Transmit request of larger N-SDU

9.4.3 Transition description

Transition	Name	Description
1	CanTp_Transmit (TxPduId, PduInfoPtr)	The PDU Router needs to transmit an I-PDU that requires transport protocol functionality and whose data refers to with the data structure information Pdu InfoPtr (see definition of type: PduInfo Type). So the PduR translates the I-PDU identifier to find which transport layer to use (CanTp, LinTp or FrTp), and what the associate N-SDU identifier is (identifier translation). Then PduR calls the CanTp's primitive CanTp_Transmit. This function shall perform these following steps: - Validates input parameters and resource availability - Searches out the useful information to process the transmit request in the configuration set of this CanTp entity (e.g. SF/FF/CF N-PDU identifier, FC N-PDU identifier, N_TA value, and so on) - Launches an internal transmit task with parameters: TxPduId and PduInfo Ptr. Note: only the length information within the PduInfoPtr structure shall be analyzed. The pointer to the payload data shall be discarded. Upon successful return of the call, the upper layer locks the required Tx buffer.
2	E_OK	The upper layer allocates and locks the required Tx buffer. Then returns E_OK.
3	PduR_CanTpCopyTxData (id, info, retry, availableDataPtr)	The PduR_CanTpCopyTxData is called. The upper layer copies segment data into the destination buffer.
4	CanIf_Transmit (TxPduId, PduInfoPtr)	Within the task, CanTp calls the CAN Interface by using CanIf_Transmit, where TxPduId identifies the L-SDU (a translation has to be preformed between the N-SDU Id used by CanTp and the L-SDU Id used by CAN Interface), and PduInfoPtr indicator data and their length.
5	CanTp_TxConfirmation(TxPduId, result)	CanTp awaits a confirmation from the CAN Interface (CanTp_Tx Confirmation)

△

Transition	Name	Description
6	PduR_CanTpCopyTxData (id, info, retry, availableDataPtr)	For each consecutive frame CanTp asks the PDU Router to provide new data to be sent.
7	PduR_CanTpTxConfirmation (id, result)	When all data have been sent, or when an error occurs, CanTp notifies PDU Router with PduR_CanTpTx Confirmation. Id identify the N-SDU which transmission is confirmed, and result indicates if transmission has been completed or not.

9.5 Large N-SDU Reception

9.5.1 Assumptions

- All input parameters are OK;
- The N-SDU data length does not fit into the associated N-PDU;
- Reception is successfully processed.

9.5.2 Sequence diagram

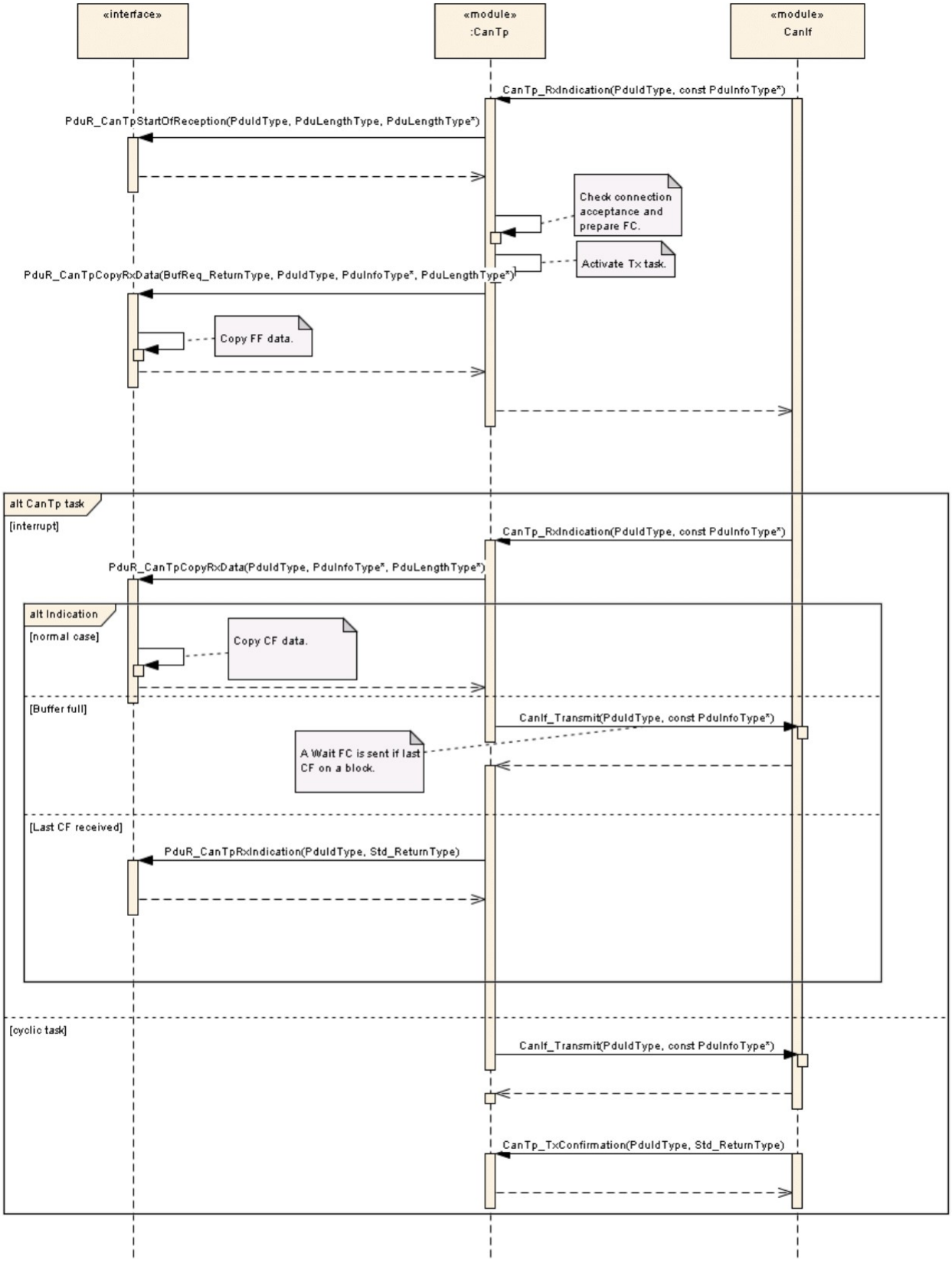

Figure 9.5: Large N-SDU Reception

Note : This sequence diagram demonstrates the working of the CAN_Tp module only. However, if the whole system is considered in such reception, more modules are involved. Since this reception can be triggered in the context of a CAN ISR, the CAN Tp operation should be as short as possible.

9.5.3 Transition description

Transition	Name	Description
1	CanTp_RxIndication (RxPduId, PduInfoPtr)	When the CAN Interface receives a frame (here a first frame), it notifies CanTp by means of a CanTp_Rx Indication callback. RxPduId represents the ID of L-PDU that has been received and PduInfoPtr indicates payload and L-SDU datalength to the L-SDU.
2	PduR_CanTpStartOfReception(id, info, TpSduLength, bufferSizePtr)	CanTp ask PDU Router to make a buffer available for incoming data with PduR_CanTpStartOfReception callback.
3		Check connection acceptance and prepare FC parameters.
4		CanTp activates a task for sending an FC with a Flow Status set to Continue ToSend. (see step 8.)
5	CanTp_RxIndication (RxPduId, PduInfoPtr)	When the CAN Interface receives a frame (here a consecutive frame), CAN Interface notifies CanTp by means of a CanTp_RxIndication callback. RxPduId represents the ID of the CAN frame that has been received and PduInfoPtr indicates payload to the L-SDU.
6		CanTp shall verify the sequence number and if correct, it asks the PduR to copy the data.
7	PduR_CanTpCopyRxData(id, info, bufferSizePtr) Or PduR_CanTpRxIndication (id, result)	Three cases can apply : - Normal Case: the received consecutive frame is not the last one. CanTp forwards received data to the upper layer. - Last CF Received: this consecutive frame is the last (Total length information was, as parameter, in the first frame). CanTp shall notify PDU Router with PduR_CanTpRxIndication callback.

▽

△

Transition	Name	Description
8		When flow control needs to be sent, the CanTp cyclical task should call the CAN Interface by using CanIf_Transmit and wait confirmation from the CAN Interface. Flow control can be either ContinueTo Send or Wait, depending on the available buffer in the upper layer.

10 Configuration specification

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

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

Chapter 10.3 specifies published information of the module CAN Transport Layer.

[SWS_CanTp_00146] ⌈The listed configuration items can be derived from a network description database, which is based on the EcuConfigurationTemplate. The configuration tool should extract all information to configure the CAN Transport Protocol.⌋ *(SRS_BSW_00159)*

[SWS_CanTp_00147] ⌈The consistency of the configuration must be checked by the configuration tool at configuration time.⌋ *(SRS_BSW_00167)*

10.1 How to read this chapter

For details refer to the chapter 10.1 "Introduction to configuration specification" in SWS_BSWGeneral.

10.2 Containers and configuration parameters

The following chapters summarize all configuration parameters. The detailed meanings of the parameters are described in Chapters 7 and 8.

[SWS_CanTp_00328] ⌈The same NPdu may only be referenced by more than one NSdu (RxNSdu or TxNSdu, via CanTpRxNPdu, CanTpTxFcNPdu, CanTpTxNPdu, or

CanTpRxFcNPdu), when either the NSdu has addressing format extended or mixed

(29 or 11 bit), or when the NPdu has MetaData.⌋ *()*

10.2.1 CanTp

Module SWS Item	ECUC_CanTp_00306	
Module Name	CanTp	
Module Description	Configuration of the CanTp (CAN Transport Protocol) module.	
Post-Build Variant Support	true	
Supported Config Variants	VARIANT-POST-BUILD, VARIANT-PRE-COMPILE	
Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
CanTpConfig	1	This container contains the configuration parameters and sub containers of the AUTOSAR CanTp module.
CanTpGeneral	1	This container contains the general configuration parameters of the CanTp module.

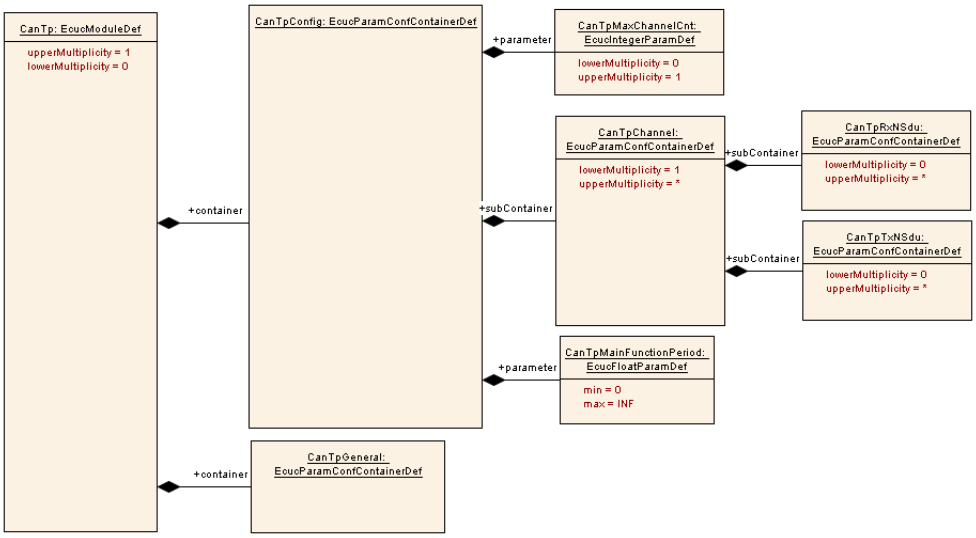

Figure 10.1: Configuration overview

10.2.2 CanTpConfig

SWS Item	[ECUC_CanTp_00290]
Container Name	CanTpConfig
Parent Container	CanTp
Description	This container contains the configuration parameters and sub containers of the AUTOSAR CanTp module.
Configuration Parameters	

Name	CanTpMainFunctionPeriod [ECUC_CanTp_00240]		
Parent Container	CanTpConfig		
Description	Allow to configure the time for the MainFunction (as float in seconds). The CanTpMainFunctionPeriod should be assigned a value which is optimal regarding all of the timers configured for CanTp in TX and RX data transfer i.e. the differences from the configured timing should be as small as possible. Please note: This period shall be the same as call cycle time of the periodic task were CanTp Main function is called.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	]0 .. INF[		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: ECU		

Name	CanTpMaxChannelCnt [ECUC_CanTp_00304]		
Parent Container	CanTpConfig		
Description	Maximum number of channels. This parameter is needed only in case of post-build loadable implementation using static memory allocation.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 18446744073709551615		
Default Value			
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
CanTpChannel	1..*	This container contains the configuration parameters of the CanTp channel.

10.2.3 CanTpGeneral

SWS Item	[ECUC_CanTp_00278]		
Container Name	CanTpGeneral		
Parent Container	CanTp		
Description	This container contains the general configuration parameters of the CanTp module.		
Configuration Parameters			

Name	CanTpChangeParameterApi [ECUC_CanTp_00299]		
Parent Container	CanTpGeneral		
Description	This parameter, if set to true, enables the CanTp_ChangeParameterRequest Api for this Module.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	CanTpDevErrorDetect [ECUC_CanTp_00239]		
Parent Container	CanTpGeneral		
Description	Switches the development error detection and notification on or off. • true: detection and notification is enabled. • false: detection and notification is disabled.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value	false		
Post-Build Variant Value	false		
Value Configuration Class	Pre-compile time	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	CanTpDynIdSupport [ECUC_CanTp_00302]
Parent Container	CanTpGeneral
Description	Enable support for dynamic ID handling via N-PDU MetaData.
Multiplicity	0..1
Type	EcucBooleanParamDef
Default Value	false
Post-Build Variant Multiplicity	false

Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	CanTpFlexibleDataRateSupport [ECUC_CanTp_00305]		
Parent Container	CanTpGeneral		
Description	Enable support for CAN FD frames.		
Multiplicity	0..1		
Type	EcucBooleanParamDef		
Default Value	true		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	CanTpGenericConnectionSupport [ECUC_CanTp_00303]		
Parent Container	CanTpGeneral		
Description	Enable support for the handling of generic connections using N-SDUs with MetaData. Requires CanTpDynIdSupport.		
Multiplicity	0..1		
Type	EcucBooleanParamDef		
Default Value	false		
Post-Build Variant Multiplicity	false		
Post-Build Variant Value	false		
Multiplicity Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local dependency: Requires CanTpDynIdSupport.		

Name	CanTpPaddingByte [ECUC_CanTp_00298]		
Parent Container	CanTpGeneral		
Description	Used for the initialization of unused bytes with a certain value		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	CanTpReadParameterApi [ECUC_CanTp_00300]		
Parent Container	CanTpGeneral		
Description	This parameter, if set to true, enables the CanTp_ReadParameterApi for this module.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	CanTpVersionInfoApi [ECUC_CanTp_00283]		
Parent Container	CanTpGeneral		
Description	The function CanTp_GetVersionInfo is configurable (On/Off) by this configuration parameter.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency			

No Included Containers

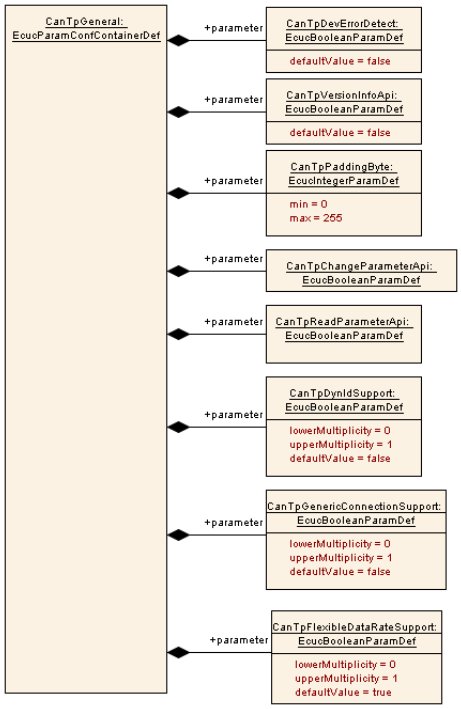

Figure 10.2: CanTpGeneral configuration overview

10.2.4 CanTpChannel

SWS Item	[ECUC_CanTp_00288]		
Container Name	CanTpChannel		
Parent Container	CanTpConfig		
Description	This container contains the configuration parameters of the CanTp channel.		
Post-Build Variant Multiplicity	true		
Multiplicity Configuration Class	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Configuration Parameters			

Included Containers		
Container Name	Multiplicity	Scope / Dependency
CanTpRxNSdu	0..*	The following parameters needs to be configured for each CAN N-SDU that the CanTp module receives via the CanTpChannel. This N-SDU produces meta data items of type SOURCE_ADDRESS_16, TARGET_ADDRESS_16 and ADDRESS_EXTENSION_8.

CanTpTxNSdu	0..*	The following parameters needs to be configured for each CAN N-SDU that the CanTp module transmits via the CanTpChannel. This N-SDU consumes meta data items of type SOURCE_ADDRESS_16, TARGET_ADDRESS_16 and ADDRESS_EXTENSION_8.

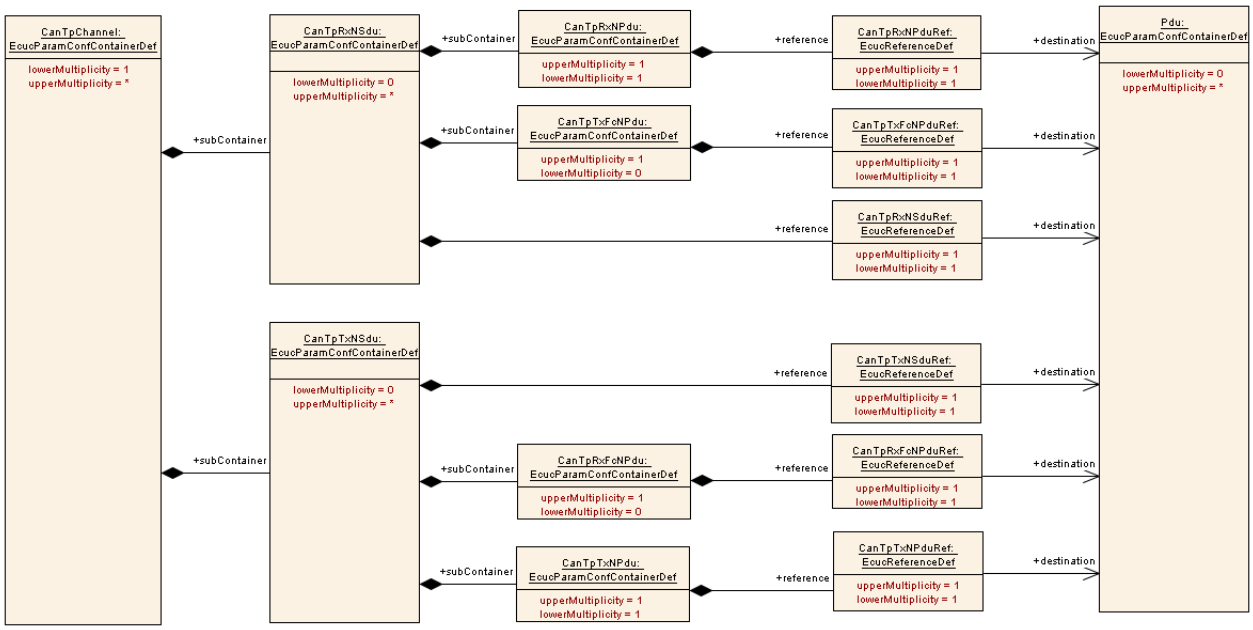

Figure 10.3: CanTpChannel configuration overview

10.2.5 CanTpRxNSdu

SWS Item	[ECUC_CanTp_00137]		
Container Name	CanTpRxNSdu		
Parent Container	CanTpChannel		
Description	The following parameters needs to be configured for each CAN N-SDU that the CanTp module receives via the CanTpChannel. This N-SDU produces meta data items of type SOURCE_ADDRESS_16, TARGET_ADDRESS_16 and ADDRESS_EXTENSION_8.		
Post-Build Variant Multiplicity	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Configuration Parameters			

Name	CanTpBs [ECUC_CanTp_00276]		
Parent Container	CanTpRxNSdu		
Description	Sets the number of N-PDUs the CanTp receiver allows the sender to send, before waiting for an authorization to continue transmission of the following N-PDUs.For further details on this parameter value see ISO 15765-2 specification.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default Value			
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Name	CanTpNar [ECUC_CanTp_00277]		
Parent Container	CanTpRxNSdu		
Description	Value in seconds of the N_Ar timeout. N_Ar is the time for transmission of a CAN frame (any N_PDU) on the receiver side.		
Multiplicity	0..1		
Type	EcucFloatParamDef		
Range	[0 .. INF]		
Default Value			
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Name	CanTpNbr [ECUC_CanTp_00245]		
Parent Container	CanTpRxNSdu		
Description	Value in seconds of the performance requirement for (N_Br + N_Ar). N_Br is the elapsed time between the receiving indication of a FF or CF or the transmit confirmation of a FC, until the transmit request of the next FC.		
Multiplicity	0..1		
Type	EcucFloatParamDef		
Range	[0 .. INF]		
Default Value			
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Name	CanTpNcr [ECUC_CanTp_00279]		
Parent Container	CanTpRxNSdu		
Description	Value in seconds of the N_Cr timeout. N_Cr is the time until reception of the next Consecutive Frame N_PDU.		
Multiplicity	0..1		
Type	EcucFloatParamDef		
Range	[0 .. INF]		
Default Value			
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Name	CanTpRxAddressingFormat [ECUC_CanTp_00281]		
Parent Container	CanTpRxNSdu		
Description	Declares which communication addressing mode is supported for this RxNSdu. Definition of Enumeration values: CanTpStandard to use normal addressing format. CanTpExtended to use extended addressing format. CanTpMixed to use mixed 11 bit addressing format. CanTpNormalFixed to use normal fixed addressing format. CanTpMixed29Bit to use mixed 29 bit addressing format.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	CANTP_EXTENDED	Extended addressing format	
	CANTP_MIXED	Mixed 11 bit addressing format	
	CANTP_MIXED29BIT	Mixed 29 bit addressing format	
	CANTP_NORMALFIXED	Normal fixed addressing format	
	CANTP_STANDARD	Normal addressing format	
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	CanTpRxNSduId [ECUC_CanTp_00301]		
Parent Container	CanTpRxNSdu		
Description	Unique identifier user by the upper layer to call CanTp_CancelReceive, CanTp_ChangeParameter and CanTp_ReadParameter.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 65535		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	CanTpRxPaddingActivation [ECUC_CanTp_00249]		
Parent Container	CanTpRxNSdu		
Description	Defines if the receive frame uses padding or not. This parameter is restricted to 8 byte N-PDUs. Definition of enumeration values: CanTpOn: The N-PDU received uses padding for SF, FC and the last CF. (N-PDU length is always >= 8 bytes in case of CAN 2.0) CanTpOff: The N-PDU received does not use padding for SF, CF and the last CF. (N-PDU length is dynamic - any valid DLC value). Note: The mandatory mapping to the next higher valid DLC value for N-PDUs with a length > 8 bytes is not affected by this parameter.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range **Post-Build Variant Value**	CANTP_OFF	Padding is not used	
	CANTP_ON	Padding is used	
	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Name	CanTpRxTaType [ECUC_CanTp_00250]		
Parent Container	CanTpRxNSdu		
Description	Declares the communication type of this Rx N-SDU.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range **Post-Build Variant Value**	CANTP_FUNCTIONAL	Functional request type	
	CANTP_PHYSICAL	Physical request type	
	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	CanTpRxWftMax [ECUC_CanTp_00251]		
Parent Container	CanTpRxNSdu		
Description	This parameter indicates how many Flow Control wait N-PDUs can be consecutively transmitted by the receiver. It is local to the node and is not transmitted inside the FC protocol data unit. CanTpRxWftMax is used to avoid sender nodes being potentially hooked-up in case of a temporarily reception inability on the part of the receiver nodes, whereby the sender could be waiting continuously.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default Value			
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Name	CanTpSTmin [ECUC_CanTp_00252]		
Parent Container	CanTpRxNSdu		
Description	Sets the duration of the minimum time the CanTp sender shall wait between the transmissions of two CF N-PDUs. For further details on this parameter value see ISO 15765-2 specification.		
Multiplicity	0..1		
Type	EcucFloatParamDef		
Range	[0 .. INF]		
Default Value			
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Name	CanTpRxNSduRef [ECUC_CanTp_00241]		
Parent Container	CanTpRxNSdu		
Description	Reference to a Pdu in the COM-Stack.		
Multiplicity	1		
Type	Reference to Pdu		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
CanTpNAe	0..1	This container is required for each RxNSdu and TxNSdu with AddressingFormat CANTP_MIXED or CANTP_MIXED29BIT.
CanTpNSa	0..1	This container is required for each RxNSdu and TxNSdu with RxTaType CANTP_PHYSICAL and CanTpAddressingFormat CANTP_EXTENDED. When DynIdSupport is enabled, this container is also required for each TxNSdu with AddressingFormat CANTP_NORMALFIXED or CANTP_MIXED29BIT. When DynIdSupport is enabled and GenericConnectionSupport is not enabled, this container is also required for each RxNSdu with AddressingFormat CANTP_NORMALFIXED or CANTP_MIXED29BIT.
CanTpNTa	0..1	This container is required for each RxNSdu and TxNSdu with AddressingFormat CANTP_EXTENDED. When DynIdSupport is enabled, this container is also required for each RxNSdu with AddressingFormat CANTP_NORMALFIXED or CANTP_MIXED29BIT. When DynIdSupport is enabled and GenericConnectionSupport is not enabled, this container is also required for each TxNSdu with AddressingFormat CANTP_NORMALFIXED or CANTP_MIXED29BIT.
CanTpRxNPdu	1	Used for grouping of the ID of a PDU and the Reference to a PDU. This N-PDU consumes a meta data item of type CAN_ID_32.
CanTpTxFcNPdu	0..1	Used for grouping of the ID of a PDU and the Reference to a PDU. This N-PDU produces a meta data item of type CAN_ID_32.

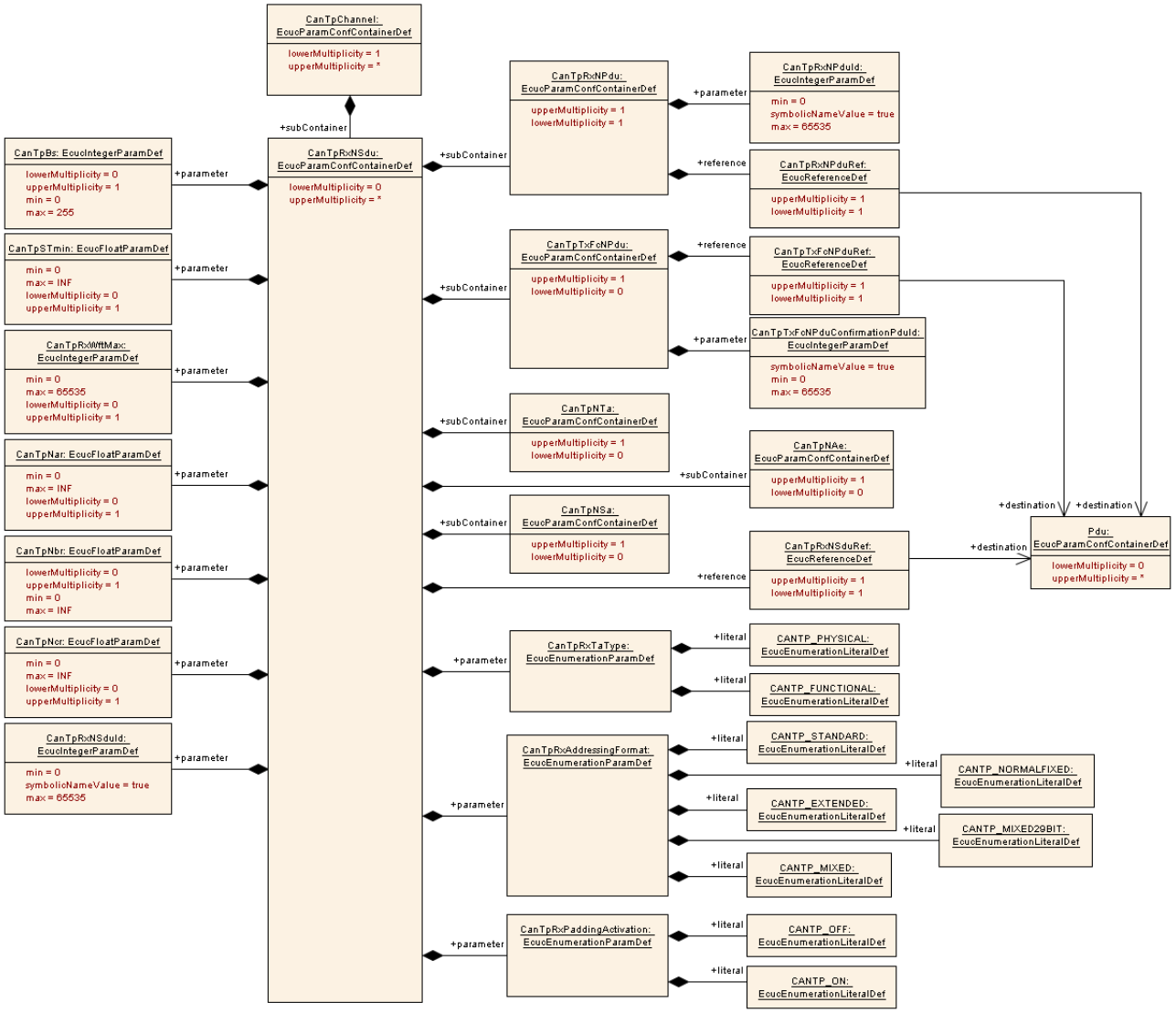

Figure 10.4: CanTpRxNSdu configuration overview

10.2.6 CanTpTxFcNPdu

SWS Item	[ECUC_CanTp_00259]
Container Name	CanTpTxFcNPdu
Parent Container	CanTpRxNSdu
Description	Used for grouping of the ID of a PDU and the Reference to a PDU. This N-PDU produces a meta data item of type CAN_ID_32.
Configuration Parameters	

Name	CanTpTxFcNPduConfirmationPduId [ECUC_CanTp_00287]		
Parent Container	CanTpTxFcNPdu		
Description	Handle Id to be used by the CanIf to confirm the transmission of the CanTpTxFcNPdu to the CanIf module.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 65535		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	CanTpTxFcNPduRef [ECUC_CanTp_00260]		
Parent Container	CanTpTxFcNPdu		
Description	Reference to a Pdu in the COM-Stack.		
Multiplicity	1		
Type	Reference to Pdu		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

No Included Containers

10.2.7 CanTpRxNPdu

SWS Item	[ECUC_CanTp_00256]
Container Name	CanTpRxNPdu
Parent Container	CanTpRxNSdu
Description	Used for grouping of the ID of a PDU and the Reference to a PDU. This N-PDU consumes a meta data item of type CAN_ID_32.
Configuration Parameters	

Name	CanTpRxNPduId [ECUC_CanTp_00258]		
Parent Container	CanTpRxNPdu		
Description	The N-PDU identifier attached to the RxNsdu is identified by CanTpRxNSduId. Each RxNsdu identifier is linked to only one SF/FF/CF N-PDU identifier. Nevertheless, in the case of extended or mixed addressing format, the same N-PDU identifier can be used for several N-SDU identifiers. The distinction is made by the N_TA or N_AE value (first data byte of SF or FF frames).		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 65535		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	CanTpRxNPduRef [ECUC_CanTp_00257]		
Parent Container	CanTpRxNPdu		
Description	Reference to a Pdu in the COM-Stack.		
Multiplicity	1		
Type	Reference to Pdu		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

No Included Containers

10.2.8 CanTpTxNSdu

SWS Item	[ECUC_CanTp_00138]
Container Name	CanTpTxNSdu
Parent Container	CanTpChannel
Description	The following parameters needs to be configured for each CAN N-SDU that the CanTp module transmits via the CanTpChannel. This N-SDU consumes meta data items of type SOURCE_ADDRESS_16, TARGET_ADDRESS_16 and ADDRESS_EXTENSION_8.
Post-Build Variant Multiplicity	true

Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Configuration Parameters			

Name	CanTpNas [ECUC_CanTp_00263]		
Parent Container	CanTpTxNSdu		
Description	Value in second of the N_As timeout. N_As is the time for transmission of a CAN frame (any N_PDU) on the part of the sender.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	[0 .. INF]		
Default Value			
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Name	CanTpNbs [ECUC_CanTp_00264]		
Parent Container	CanTpTxNSdu		
Description	Value in seconds of the N_Bs timeout. N_Bs is the time of transmission until reception of the next Flow Control N_PDU.		
Multiplicity	0..1		
Type	EcucFloatParamDef		
Range	[0 .. INF]		
Default Value			
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Name	CanTpNcs [ECUC_CanTp_00265]		
Parent Container	CanTpTxNSdu		
Description	Value in seconds of the performance requirements relating to N_Cs. CanTpNcs is the time in which CanTp is allowed to request from PduR the Tx data of a Consecutive Frame N_PDU.		
Multiplicity	0..1		
Type	EcucFloatParamDef		
Range	[0 .. INF]		
Default Value			
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local		

Name	CanTpTc [ECUC_CanTp_00282]		
Parent Container	CanTpTxNSdu		
Description	Switch for enabling Transmit Cancellation.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	CanTpTxAddressingFormat [ECUC_CanTp_00262]	
Parent Container	CanTpTxNSdu	
Description	Declares which communication addressing format is supported for this TxNSdu. Definition of Enumeration values: CanTpStandard to use normal addressing format. CanTpExtended to use extended addressing format. CanTpMixed to use mixed 11 bit addressing format. CanTpNormalFixed to use normal fixed addressing format. CanTpMixed29Bit to use mixed 29 bit addressing format.	
Multiplicity	1	
Type	EcucEnumerationParamDef	
Range	CANTP_EXTENDED	Extended addressing format
	CANTP_MIXED	Mixed 11 bit addressing format
	CANTP_MIXED29BIT	Mixed 29 bit addressing format

	CANTP_NORMALFIXED	Normal fixed addressing format	
	CANTP_STANDARD	Normal addressing format	
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	CanTpTxNSduId [ECUC_CanTp_00268]		
Parent Container	CanTpTxNSdu		
Description	Unique identifier to a structure that contains all useful information to process the transmission of a TxNsdu.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 65535		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	CanTpTxPaddingActivation [ECUC_CanTp_00269]		
Parent Container	CanTpTxNSdu		
Description	Defines if the transmit frame use padding or not. This parameter is restricted to 8 byte N-PDUs. Definition of Enumeration values: CanTpOn The transmit N-PDU uses padding for SF, FC and the last CF. (N-PDU length is always 8 bytes in case of CAN 2.0) CanTpOff The transmit N-PDU does not use padding for SF, CF and the last CF. (N-PDU length is dynamic - any valid DLC value). Note: The mandatory mapping to the next higher valid DLC value for N-PDUs with a length > 8 bytes is not affected by this parameter.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	CANTP_OFF	Padding is not used	
	CANTP_ON	Padding is used	
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD

Scope / Dependency	scope: local

Name	CanTpTxTaType [ECUC_CanTp_00270]		
Parent Container	CanTpTxNSdu		
Description	Declares the communication type of this TxNsdu. Enumeration values: CanTpPhysical. Used for 1:1 communication. CanTpFunctional. Used for 1:n communication.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	CANTP_FUNCTIONAL	Functional request type	
	CANTP_PHYSICAL	Physical request type	
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	CanTpTxNSduRef [ECUC_CanTp_00261]		
Parent Container	CanTpTxNSdu		
Description	Reference to a Pdu in the COM-Stack.		
Multiplicity	1		
Type	Reference to Pdu		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
CanTpNAe	0..1	This container is required for each RxNSdu and TxNSdu with AddressingFormat CANTP_MIXED or CANTP_MIXED29BIT.
CanTpNSa	0..1	This container is required for each RxNSdu and TxNSdu with RxTaType CANTP_PHYSICAL and CanTpAddressingFormat CANTP_EXTENDED. When DynIdSupport is enabled, this container is also required for each TxNSdu with AddressingFormat CANTP_NORMALFIXED or CANTP_MIXED29BIT. When DynIdSupport is enabled and GenericConnectionSupport is not enabled, this container is also required for each RxNSdu with AddressingFormat CANTP_NORMALFIXED or CANTP_MIXED29BIT.

CanTpNTa	0..1	This container is required for each RxNSdu and TxNSdu with AddressingFormat CANTP_EXTENDED. When DynIdSupport is enabled, this container is also required for each RxNSdu with AddressingFormat CANTP_NORMALFIXED or CANTP_MIXED29BIT. When DynIdSupport is enabled and GenericConnectionSupport is not enabled, this container is also required for each TxNSdu with AddressingFormat CANTP_NORMALFIXED or CANTP_MIXED29BIT.
CanTpRxFcNPdu	0..1	Used for grouping of the ID of a PDU and the Reference to a PDU. This N-PDU consumes a meta data item of type CAN_ID_32.
CanTpTxNPdu	1	Used for grouping of the ID of a PDU and the Reference to a PDU. This N-PDU produces a meta data item of type CAN_ID_32.

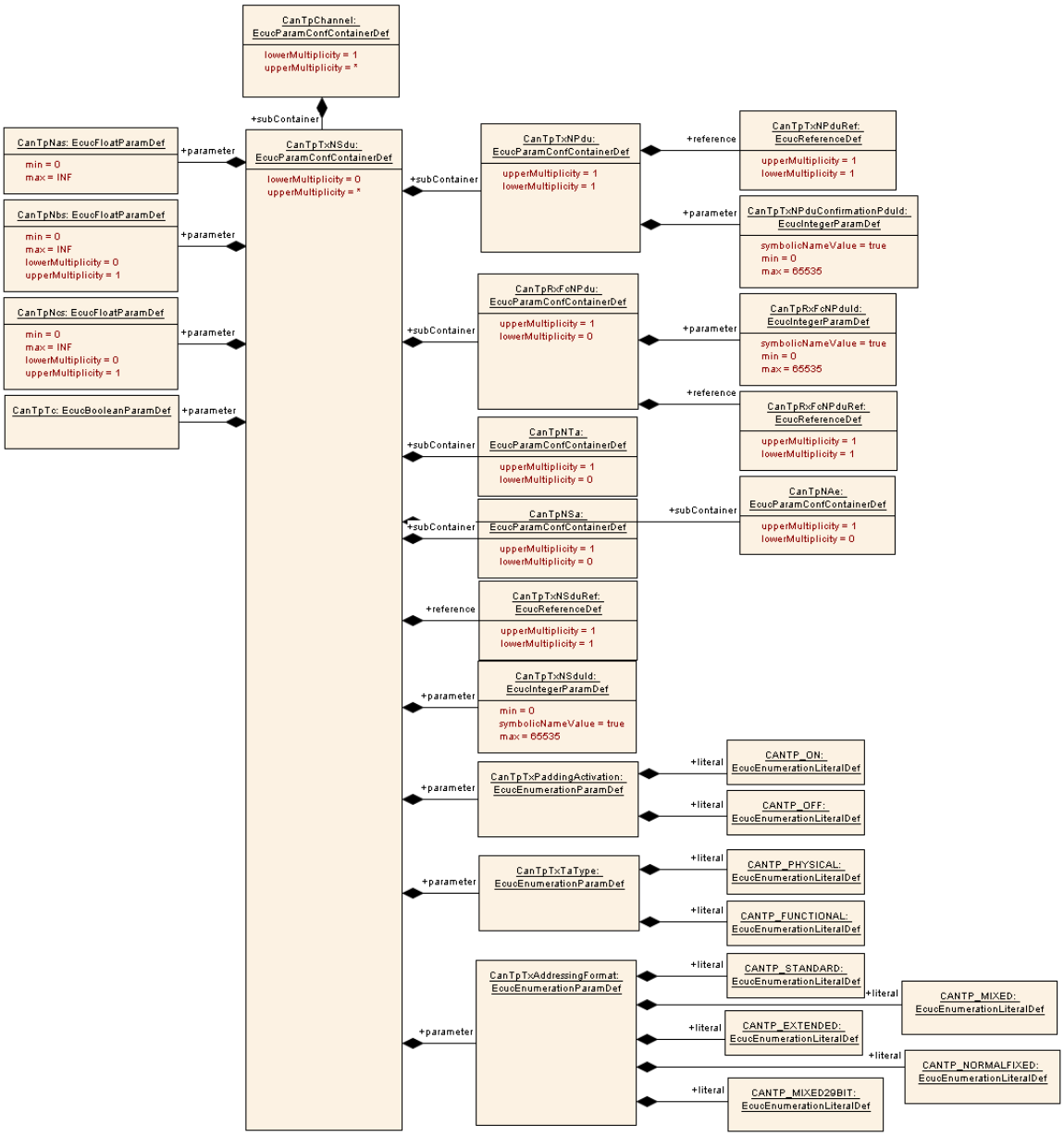

Figure 10.5: CanTpTxNSdu configuration overview

10.2.9 CanTpTxNPdu

SWS Item	[ECUC_CanTp_00274]
Container Name	CanTpTxNPdu
Parent Container	CanTpTxNSdu
Description	Used for grouping of the ID of a PDU and the Reference to a PDU. This N-PDU produces a meta data item of type CAN_ID_32.
Configuration Parameters	

Name	CanTpTxNPduConfirmationPduId [ECUC_CanTp_00286]		
Parent Container	CanTpTxNPdu		
Description	Handle Id to be used by the CanIf to confirm the transmission of the CanTpTxNPdu to the CanIf module.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 65535		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	CanTpTxNPduRef [ECUC_CanTp_00275]		
Parent Container	CanTpTxNPdu		
Description	Reference to a Pdu in the COM-Stack.		
Multiplicity	1		
Type	Reference to Pdu		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

No Included Containers

10.2.10 CanTpRxFcNPdu

SWS Item	[ECUC_CanTp_00271]
Container Name	CanTpRxFcNPdu
Parent Container	CanTpTxNSdu
Description	Used for grouping of the ID of a PDU and the Reference to a PDU. This N-PDU consumes a meta data item of type CAN_ID_32.
Configuration Parameters	

Name	CanTpRxFcNPduId [ECUC_CanTp_00273]		
Parent Container	CanTpRxFcNPdu		
Description	N-PDU identifier attached to the FC N-PDU of this TxNsdu identified by CanTpTxNSduId. Each TxNsdu identifier is linked to one Rx FC N-PDU identifier only. However, in the case of extended addressing format, the same FC N-PDU identifier can be used for several N-SDU identifiers. The distinction is made by means of the N_TA value (first data byte of FC frames).		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 65535		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

Name	CanTpRxFcNPduRef [ECUC_CanTp_00272]		
Parent Container	CanTpRxFcNPdu		
Description	Reference to a Pdu in the COM-Stack.		
Multiplicity	1		
Type	Reference to Pdu		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	–	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

No Included Containers

10.2.11 CanTpNTa

SWS Item	[ECUC_CanTp_00139]
Container Name	CanTpNTa
Parent Container	CanTpRxNSdu, CanTpTxNSdu

Description	This container is required for each RxNSdu and TxNSdu with AddressingFormat CANTP_EXTENDED. When DynIdSupport is enabled, this container is also required for each RxNSdu with AddressingFormat CANTP_NORMALFIXED or CANTP_MIXED29BIT. When DynIdSupport is enabled and GenericConnectionSupport is not enabled, this container is also required for each TxNSdu with AddressingFormat CANTP_NORMALFIXED or CANTP_MIXED29BIT.
Configuration Parameters	

Name	CanTpNTa [ECUC_CanTp_00255]		
Parent Container	CanTpNTa		
Description	This parameter contains the transport protocol target address value.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

No Included Containers

10.2.12 CanTpNSa

SWS Item	[ECUC_CanTp_00253]
Container Name	CanTpNSa
Parent Container	CanTpRxNSdu, CanTpTxNSdu
Description	This container is required for each RxNSdu and TxNSdu with RxTaType CANTP_PHYSICAL and CanTpAddressingFormat CANTP_EXTENDED. When DynIdSupport is enabled, this container is also required for each TxNSdu with AddressingFormat CANTP_NORMALFIXED or CANTP_MIXED29BIT. When DynIdSupport is enabled and GenericConnectionSupport is not enabled, this container is also required for each RxNSdu with AddressingFormat CANTP_NORMALFIXED or CANTP_MIXED29BIT.
Configuration Parameters	

Name	CanTpNSa [ECUC_CanTp_00254]	
Parent Container	CanTpNSa	
Description	This parameter contains the transport protocol source address value.	
Multiplicity	1	
Type	EcucIntegerParamDef	
Range	0 .. 255	

Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

No Included Containers

10.2.13 CanTpNAe

SWS Item	[ECUC_CanTp_00284]
Container Name	CanTpNAe
Parent Container	CanTpRxNSdu, CanTpTxNSdu
Description	This container is required for each RxNSdu and TxNSdu with AddressingFormat CANTP_MIXED or CANTP_MIXED29BIT.
Configuration Parameters	

Name	CanTpNAe [ECUC_CanTp_00285]		
Parent Container	CanTpNAe		
Description	This parameter contains the transport protocol address extension value.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default Value			
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local		

No Included Containers

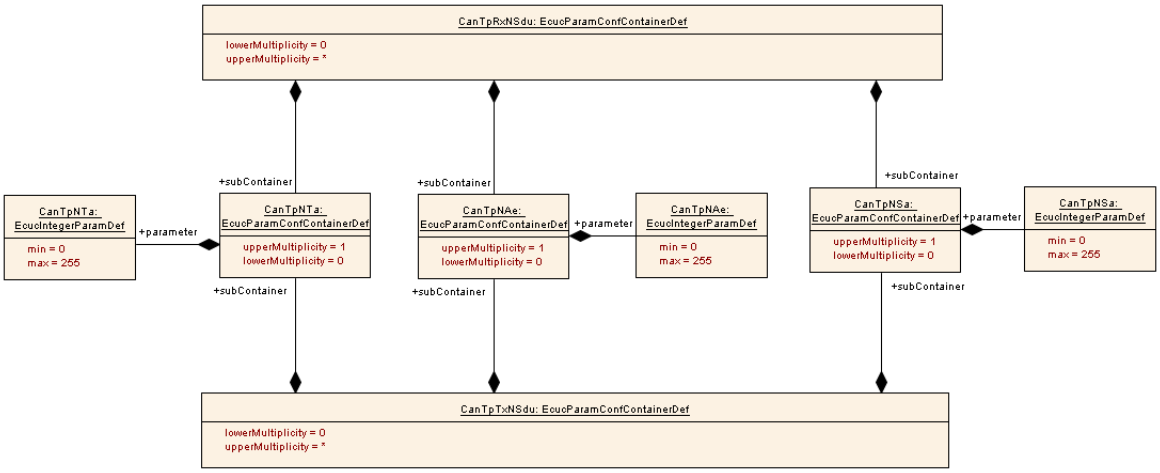

Figure 10.6: CanTpNTa, CanTpNSa and CanTpNAe configuration overview

10.3 Published Information

For details refer to the chapter 10.3 "Published Information" in SWS_BSWGeneral.

A Not applicable requirements

[SWS_CanTp_00327] ⌈These requirements are not applicable to this specification.⌋ *(SRS_BSW_00344, SRS_BSW_00404, SRS_BSW_00405, SRS_BSW_00170, SRS_BSW_00419, SRS_BSW_00383, SRS_BSW_00397, SRS_BSW_00398, SRS_BSW_00399, SRS_BSW_00400, SRS_BSW_00375, SRS_BSW_00416, SRS_BSW_00168, SRS_BSW_00423, SRS_BSW_00427, SRS_BSW_00428, SRS_BSW_00429, SRS_BSW_00432, SRS_BSW_00433, SRS_BSW_00422, SRS_BSW_00417, SRS_BSW_00161, SRS_BSW_00162, SRS_BSW_00415, SRS_BSW_00325, SRS_BSW_00342, SRS_BSW_00413, SRS_BSW_00347, SRS_BSW_00307, SRS_BSW_00314, SRS_BSW_00361, SRS_BSW_00328, SRS_BSW_00378, SRS_BSW_00172, SRS_BSW_00010, SRS_BSW_00321, SRS_BSW_00341, SRS_BSW_00334)*