Document Title	Specification of CAN Transport Layer
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	014
Document Classification	Standard

Document Version	4.0.0
Document Status	Final
Part of Release	4.0
Revision	3

Document Change History

Date	Version	Changed by	Change Description
01.12.2011	4.0.0	AUTOSAR Administration	• 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
22.10.2010	3.1.0	AUTOSAR Administration	• 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
07.12.2009	3.0.0	AUTOSAR Administration	• Added Mixed Addressing Mode • CanTp supports Full Duplex Mode • New buffering concept • Added possibility to change CanTp parameters • Legal disclaimer revised
23.06.2008	2.2.1	AUTOSAR Administration	• Legal disclaimer revised
03.12.2007	2.2.0	AUTOSAR Administration	• Addition of transmit cancellation feature • DataLength check only for too small DLC (CanTp220) • Restriction on mapping of N-Pdu (CanTp248) • Document meta information extended • Small layout adaptations made

Document Change History			
Date	**Version**	**Changed by**	**Change Description**
24.01.2007	2.1.1	AUTOSAR Administration	• "Advice for users" revised • "Revision Information" added
04.12.2006	2.1.0	AUTOSAR Administration	• Clarification and correction of error management: list of production/development error and behavior in case of error • Addition of CanTp166 and CanTp167 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
27.04.2006	2.0.0	AUTOSAR Administration	Document structure adapted to common Release 2.0 SWS Template.
21.06.2005	1.0.0	AUTOSAR Administration	Initial Release

Disclaimer

Advice for users

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

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

Table of Contents

1 Introduction and functional overview

This specification 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). The main purpose of the CAN TP module is to segment and reassemble CAN I-PDUs longer than 8 bytes.
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 FlexRay) 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.

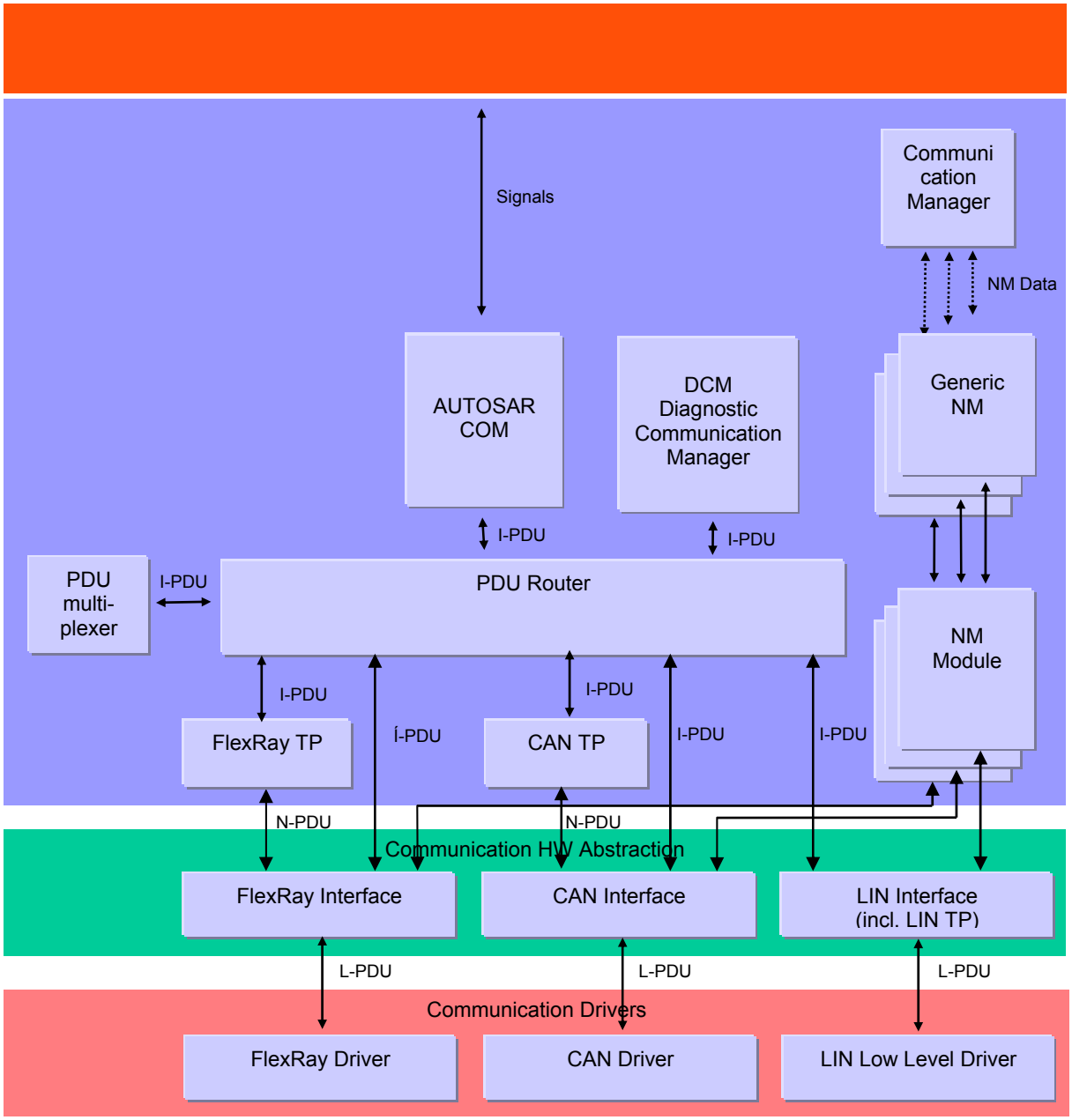


Figure 1 : AUTOSAR Communication Stack

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 [13] and ISO 15765-4 for OBD diagnostics [15]. 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 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.
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

Abbreviation:	Description:
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	Development 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).
N_As	Time for transmission of the CAN frame (any N-PDU) on the sender side (see ISO 15765-2).
N_Br	Time until transmission of the next flow control N-PDU (see ISO 15765-2).
N_Bs	Time until reception of the next flow control N-PDU (see ISO 15765-2).
N_Cr	Time until reception of the next consecutive frame N-PDU (see ISO 15765-2).
N_Cs	Time until transmission of the next consecutive frame N-PDU (see ISO 15765-2).
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).
N_TA	Network Target Address (see ISO 15765-2). It might already contain the N_TAtype(physical/function) in case of ExtendedAddressing.
N_TAtype	Network Target Address type (see ISO 15765-2).
OBD	On-Board Diagnostic
PDU	Protocol Data Unit
PduR	PDU Router
SDU	Service Data Unit

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

Definitions:	Description:
Development Error Tracer	The Development 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.
Full-duplex	Point-to-point communication between two nodes is possible in both directions at any one time.

Definitions:	*Description:*
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 [13] 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 “CommunicationControl” OBD communication will always be performed as part of functional addressing.
Half-duplex	Point-to-point communication between two nodes is only possible in one direction at a time.
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. Please note that the current specification is intended to support only the 11 bit CAN identifier Mixed addressing format.
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.
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 [13] 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 “InputOutputControlByIdentifier”
Single connection	The CAN Transport Layer will only manage one transport protocol communication session at a time.

3 Related documentation

3.1 Input documents

[1] List of Basic Software Modules,
AUTOSAR_TR_BSWModuleList.pdf

[2] Layered Software Architecture,
AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

[3] General Requirements on Basic Software Modules,
AUTOSAR_SRS_BSWGeneral.pdf

[4] Specification of ECU Configuration,
AUTOSAR_TPS_ECUConfiguration.pdf

[5] Glossary
AUTOSAR_TR_Glossary.pdf

[6] Requirements on CAN
AUTOSAR_SRS_CAN.pdf

[7] Specification of CAN Interface
AUTOSAR_SWS_CANInterface.pdf

[8] API Specification of Development Error Tracer
AUTOSAR_SWS_DevelopmentErrorTracer.pdf

[9] Specification of Function Inhibition Manager
AUTOSAR_SWS_FunctionInhibitionManager.pdf

[10] Specification of PDU Router
AUTOSAR_SWS_PDURouter.pdf

[11] Specification of Diagnostic Event Manager
AUTOSAR_SWS_DiagnosticEventManager.pdf

[12] Basic Software Module Description Template,
AUTOSAR_TPS_BSWModuleDescriptionTemplate.pdf

3.2 Related standards and norms

[13] ISO 15765-2 (2004-10-12), Road vehicles — Diagnostics on Controller Area Networks (CAN) — Part2: Network layer services

[14] ISO 15765-3 (2004-10-06), Road vehicles — Diagnostics on Controller Area Networks (CAN) — Part3: Implementation of diagnostic services

[15] ISO 15765-4 (2005-01-04), Road vehicles — Diagnostics on Controller Area Networks (CAN) — Part4: Requirements for emissions-related systems

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

This CAN Transport Layer implementation supports half and full-duplex communication; support for full-duplex communication is configurable on channel base (see Chapter 10).

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 on 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).

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.

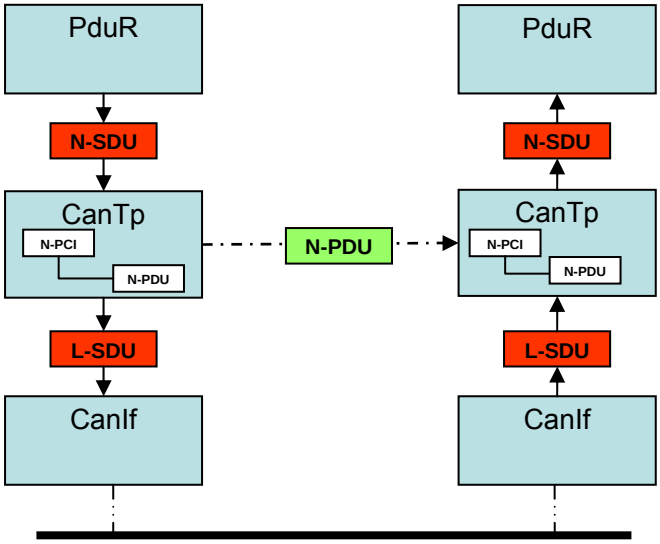


Figure 2: CAN Transport Layer interactions

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 buffer allocation request until the reception indication or the next buffer allocation request 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)
- Communication type on each channel – half or full-duplex

- Addressing type of each CAN N-SDU (physical or functional)
- Addressing format of each connection (standard, 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
- Minimum length of the N-SDU

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 declares and requests certain callback functions to confirm transmission, confirm transmission cancellation and notify reception of a message from/to the PDU-Router, and request a buffer, to reassemble segmented frames:

- *PduR_CanTpRxIndication*
- *PduR_CanTpStartOfReception*
- *PduR_CanTpCopyRxData*
- *PduR_CanTpCopyTxData*
- *PduR_CanTpTxConfirmation*

For more information about these functions, refer to the PDU Router module specification [10].

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 [7].

5.2 File structure

5.2.1 Code file structure

[CANTP159] ⌈The code file structure will not be completely defined within this specification. The CanTp module code file structure should include CanTp_Lcfg.c containing the link time configurable parameters.⌋ (BSW380)

[CANTP259] ⌈The CanTp module code file structure should include CanTp_PBcfg.c containing the post-build time configurable parameters.

These files will contain all link time and post-build configurable parameters. ⌋( )

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.

The header file structure defined in this section allows the separation of platform, compiler and implementation specific definitions and declarations from general definitions, as well as the separation of source code and configuration.

[CANTP156] ⌈The CanTp module shall implement the file structure as shown in Figure 3.

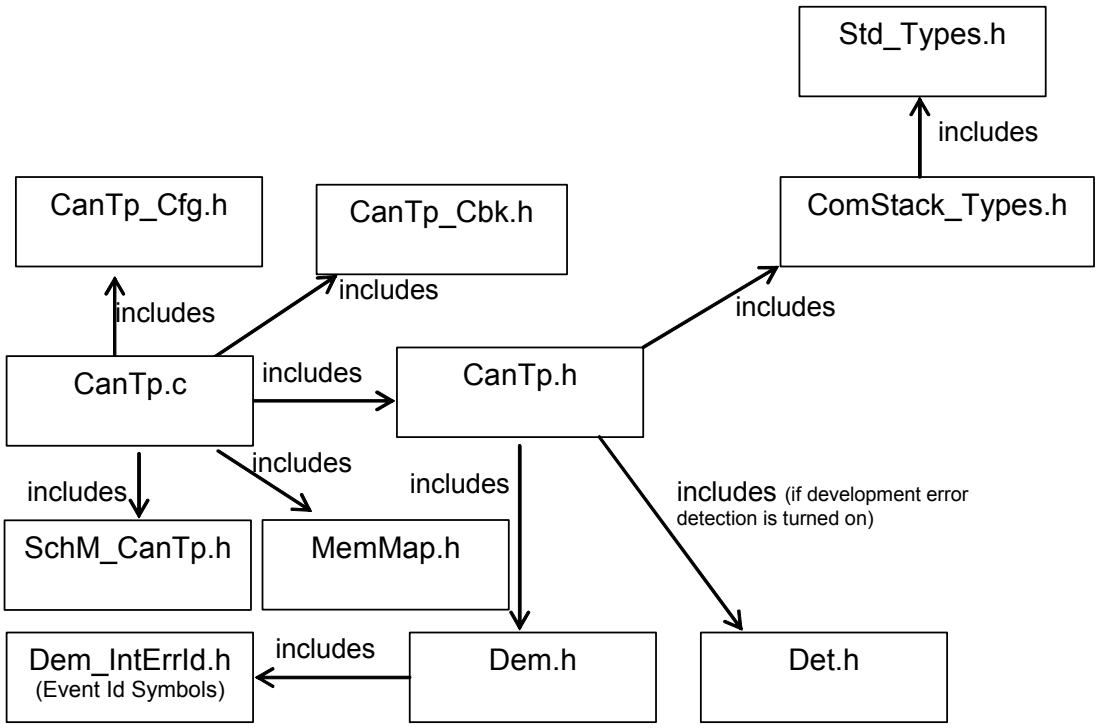


Figure 3: File Structure

⌋(BSW412, BSW00435, BSW00436, BSW00346, BSW158, BSW00370, BSW00301)

[CANTP265] ⌈Global data types and functions that are only used internally by the CAN Transport Protocol, are given in CanTp.c. ⌋( )

[CANTP219] ⌈File CanTp.c shall include CanTp_Cfg.h, CanTp_Cbk.h, CanTp.h, MemMap.h and SchM_CanTp.h. ⌋( )

[CANTP264] ⌈File CanTp.h shall include ComStack_Types.h and Det.h (optional, if development error detection is configured as ON). ⌋( )

[CANTP157] ⌈The file CanTp.h shall only contain 'external' declarations of constants, global data, type definitions and services that are specified in the CAN Transport Protocol SWS. ⌋(BSW00302)

[CANTP001] ⌈CanTp_Cfg.h shall define constant and customizable data for module configuration at pre-compile time. ⌋(BSW00345, BSW381, BSW158)

[CANTP221] ⌈CanTp.c shall include CanTp_Cfg.h. ⌋( )

[CANTP165] ⌈BSW scheduler information is included via SchM_CanTp.h. ⌋( )

[CANTP160] ⌈References to c-configuration parameters (link time and post-build time) will be placed in a separate h-file. The h-file should be the same as pre-compilation parameters. ⌋( )

5.2.3 Version check

[CANTP266] ⌈Header file of the CanTp module shall provide the possibility of version identification of the CAN Transport module by the following macrodefinitions:

CANTP_SW_MAJOR_VERSION,
CANTP_SW_MINOR_VERSION,
CANTP_SW_PATCH_VERSION,
CANTP_AR_ RELEASE_MAJOR_VERSION,
CANTP_AR_ RELEASE_MINOR_VERSION,
CANTP_AR_ RELEASE_REVISION_VERSION.
⌋( )

[CANTP024] ⌈Source file of the CanTp module shall provide the possibility of version identification of the CAN Transport module by CANTP_SW_MAJOR_VERSION, CANTP_SW_MINOR_VERSION and CANTP_SW_PATCH_VERSION. ⌋(BSW004, BSW003)

[CANTP307] ⌈The CanTp module shall perform Inter Module Checks to avoid integration of incompatible files. The imported included files shall be checked by preprocessing directives.
The following version numbers shall be verified:
- <MODULENAME>_AR_RELEASE_MAJOR_VERSION
- <MODULENAME>_AR_RELEASE_MINOR_VERSION,
where <MODULENAME> is the module abbreviation of the other (external) modules which provide header files included by the CanTp module. ⌋( )

[CANTP308] ⌈If the values are not identical to the expected values, an error shall be reported.⌋( )

[CANTP267] ⌈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

[CANTP150] ⌈The CanTp module's source (as long as it is written in C) shall conform to the HIS subset of the MISRA C Standard.⌋(BSW007)

[CANTP151] ⌈The CanTp module's source shall not use compiler and platform specific keywords⌋(BSW00306)

[CANTP152] ⌈The CanTp module's source shall indicate all global data with read-only properties by explicitly assigning the keyword `const`.⌋(BSW00309)

[CANTP153] ⌈The CanTp module may use macros (instead of functions) where source code is used and runtime is critical.⌋(BSW00330)

[CANTP155] ⌈The CanTp module shall not define global data in header files (If global variables have to be used, the definition should take place in the C file)⌋(BSW00308)

[CANTP158] ⌈The CanTp module's source shall not be processor and compiler dependent.⌋(BSW006)

6 Requirements traceability

Requirement	Satisfied by
-	CANTP168
-	CANTP211
-	CANTP217
-	CANTP296
-	CANTP249
-	CANTP283
-	CANTP027
-	CANTP223
-	CANTP299
-	CANTP319
-	CANTP271
-	CANTP302
-	CANTP293
-	CANTP222
-	CANTP279
-	CANTP238
-	CANTP281
-	CANTP266
-	CANTP087
-	CANTP318
-	CANTP268
-	CANTP076
-	CANTP081
-	CANTP300
-	CANTP095
-	CANTP226
-	CANTP166
-	CANTP176
-	CANTP306
-	CANTP269
-	CANTP320
-	CANTP280
-	CANTP209
-	CANTP272
-	CANTP265
-	CANTP312
-	CANTP262
-	CANTP086

-	CANTP225
-	CANTP111
-	CANTP218
-	CANTP321
-	CANTP307
-	CANTP206
-	CANTP229
-	CANTP082
-	CANTP316
-	CANTP075
-	CANTP134
-	CANTP288
-	CANTP264
-	CANTP084
-	CANTP323
-	CANTP200
-	CANTP313
-	CANTP285
-	CANTP294
-	CANTP263
-	CANTP267
-	CANTP216
-	CANTP114
-	CANTP311
-	CANTP233
-	CANTP261
-	CANTP167
-	CANTP115
-	CANTP221
-	CANTP250
-	CANTP202
-	CANTP284
-	CANTP199
-	CANTP325
-	CANTP304
-	CANTP270
-	CANTP093
-	CANTP248
-	CANTP253
-	CANTP215
-	CANTP259
-	CANTP273

-	CANTP246
-	CANTP212
-	CANTP243
-	CANTP184
-	CANTP277
-	CANTP210
-	CANTP067
-	CANTP224
-	CANTP309
-	CANTP310
-	CANTP289
-	CANTP090
-	CANTP236
-	CANTP303
-	CANTP133
-	CANTP094
-	CANTP314
-	CANTP282
-	CANTP185
-	CANTP165
-	CANTP324
-	CANTP298
-	CANTP255
-	CANTP160
-	CANTP322
-	CANTP274
-	CANTP231
-	CANTP244
-	CANTP092
-	CANTP232
-	CANTP251
-	CANTP080
-	CANTP308
-	CANTP079
-	CANTP256
-	CANTP254
-	CANTP064
-	CANTP287
-	CANTP190
-	CANTP242
-	CANTP205
-	CANTP278

-	CANTP315
-	CANTP177
-	CANTP204
-	CANTP193
-	CANTP305
-	CANTP260
-	CANTP219
-	CANTP091
-	CANTP235
-	CANTP089
-	CANTP252
-	CANTP286
-	CANTP257
-	CANTP317
-	CANTP169
-	CANTP291
-	CANTP214
-	CANTP297
-	CANTP078
-	CANTP213
BSW003	CANTP024
BSW00301	CANTP156
BSW00302	CANTP157
BSW00306	CANTP151
BSW00307	CANTP327
BSW00308	CANTP155
BSW00309	CANTP152
BSW00310	CANTP003
BSW00314	CANTP327
BSW00321	CANTP327
BSW00323	CANTP132
BSW00324	CANTP327
BSW00325	CANTP327
BSW00326	CANTP327
BSW00327	CANTP101
BSW00328	CANTP327
BSW00330	CANTP153
BSW00331	CANTP101
BSW00334	CANTP327
BSW00336	CANTP010
BSW00337	CANTP101
BSW00339	CANTP008

BSW00341	CANTP327
BSW00342	CANTP327
BSW00344	CANTP327
BSW00345	CANTP001
BSW00346	CANTP156
BSW00347	CANTP327
BSW00350	CANTP006
BSW00353	CANTP002
BSW00358	CANTP208
BSW00361	CANTP327
BSW00369	CANTP021
BSW00370	CANTP156
BSW00373	CANTP164
BSW00375	CANTP327
BSW00376	CANTP164
BSW00378	CANTP327
BSW004	CANTP024
BSW00404	CANTP327
BSW00405	CANTP327
BSW00413	CANTP327
BSW00414	CANTP208
BSW00415	CANTP327
BSW00416	CANTP327
BSW00417	CANTP327
BSW00419	CANTP327
BSW00420	CANTP327
BSW00422	CANTP327
BSW00423	CANTP327
BSW00424	CANTP164
BSW00427	CANTP327
BSW00428	CANTP327
BSW00429	CANTP327
BSW00431	CANTP327
BSW00432	CANTP327
BSW00433	CANTP327
BSW00434	CANTP327
BSW00435	CANTP156
BSW00436	CANTP156
BSW006	CANTP158
BSW007	CANTP150
BSW010	CANTP327
BSW01065	CANTP033

BSW01066	CANTP120, CANTP124, CANTP123, CANTP122, CANTP121, CANTP096
BSW01068	CANTP035
BSW01069	CANTP035
BSW01071	CANTP035
BSW01073	CANTP116, CANTP040, CANTP098
BSW01075	CANTP170, CANTP030
BSW01076	CANTP031
BSW01078	CANTP035
BSW01082	CANTP057
BSW01086	CANTP059
BSW01117	CANTP057, CANTP290
BSW01149	CANTP057, CANTP290
BSW101	CANTP208
BSW158	CANTP156, CANTP001
BSW161	CANTP327
BSW162	CANTP327
BSW168	CANTP327
BSW170	CANTP327
BSW172	CANTP327
BSW380	CANTP159
BSW381	CANTP001
BSW382	CANTP327
BSW383	CANTP327
BSW385	CANTP101
BSW386	CANTP101
BSW397	CANTP327
BSW398	CANTP327
BSW399	CANTP327
BSW400	CANTP327
BSW406	CANTP161
BSW407	CANTP162, CANTP163
BSW412	CANTP156

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 [13]) and should be able to fulfill ISO 15765-4 (Requirements for emissions-related systems [15]).

[CANTP033] ⌈If a recommendation of ISO 15765-2 is not explicitly excluded in the SWS, the CanTp module shall follow this recommendation.⌋(BSW01065)

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 [13].

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

[CANTP027] ⌈The CanTp module shall have two internal states, `CANTP_OFF` and `CANTP_ON`.⌋( )

[CANTP253] ⌈The `CANTP_OFF` and `CANTP_ON` states shall be available for debugging (chapter 7.7 details implementation of debugging concept).⌋( )

[CANTP168] ⌈The CanTp module shall be in the CANTP_OFF state after power up.⌋ ( )

[CANTP169] ⌈In the state CANTP_OFF, the CanTp shall allow an update of the postbuild configuration.⌋ ( )

[CANTP170] ⌈The CanTp module shall change to the internal state CANTP_ON when the CanTp has been successfully initialized with CanTp_Init().⌋ (BSW01075)

[CANTP238] ⌈The CanTp module shall perform segmentation and reassembly tasks only when the CanTp is in the CANTP_ON state.⌋ ( )

[CANTP030] ⌈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). ⌋ (BSW01075)

[CANTP031] ⌈If development error detection for the CanTp module is enabled the CanTp module shall raise an error (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.⌋ (BSW01076)

[CANTP111] ⌈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).⌋ ( )

[CANTP273] ⌈The CanTp module shall loose all current connections if CanTp_Init is called when CanTp module is in the global state CANTP_ON.⌋ ( )

[CANTP010] ⌈The function CanTp_Shutdown shall stop the CanTp module properly.⌋ (BSW00336)

The following figure summarizes all of the above requirements:

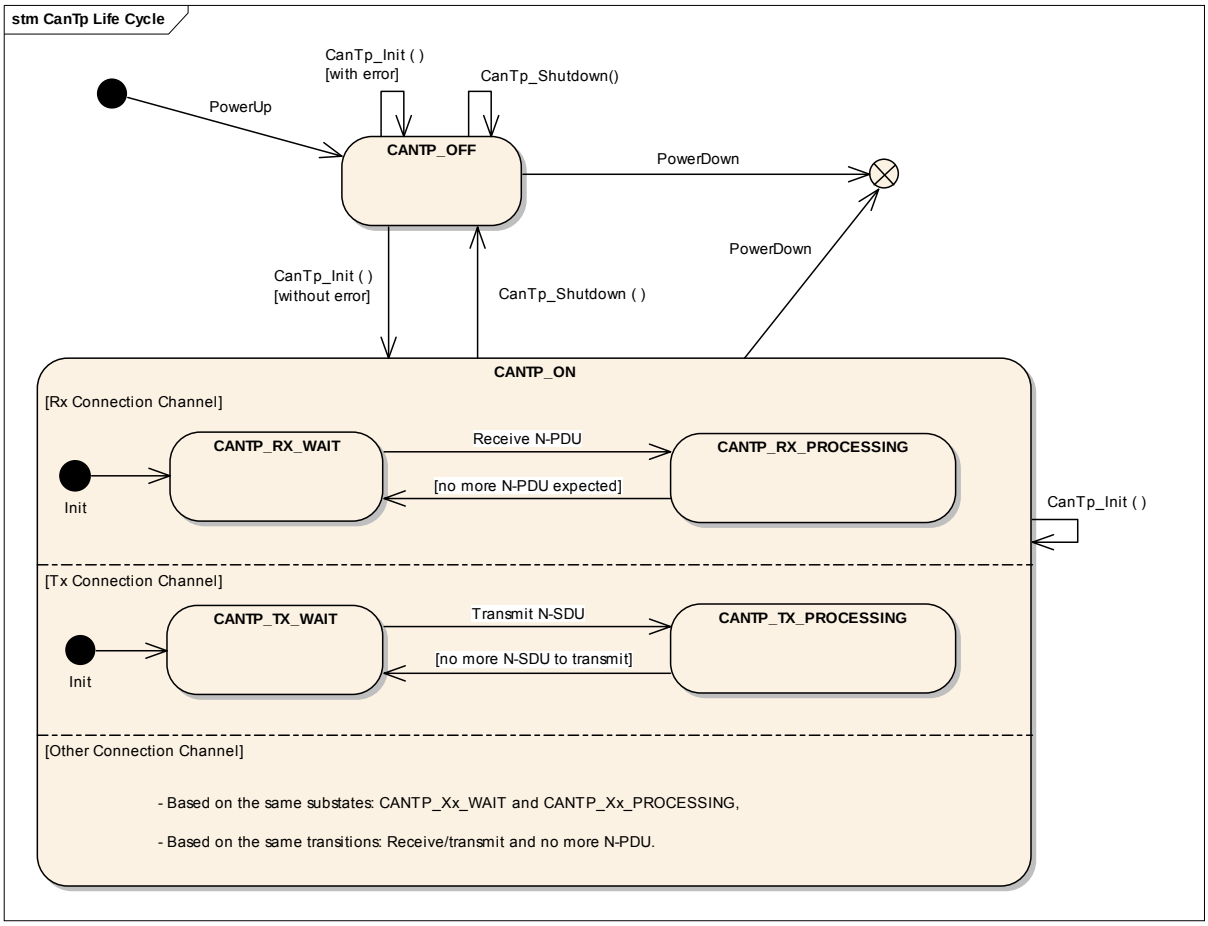


Figure 4: CAN Transport Layer life cycle

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).

[CANTP176] ⌈The function CanTp_Transmit() shall be asynchronous.⌋ ( )

[CANTP177]⌈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.

[CANTP242] ⌈This feature shall be (de)activated by static configuration (parameter CanTpTc). ⌋( )

[CANTP274] ⌈Transmit Cancellation is triggered by the call of CanTp_CancelTransmit().⌋( )

[CANTP243] ⌈After the call of the service CanTp_CancelTransmit(), the transfer on this connection shall be aborted. ⌋( )

[CANTP244] ⌈The Api PduR_CanTpTxConfirmation() shall be called after a transmit cancellation with value NTFRSLT_E_CANCELATION_OK. ⌋( )

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 performed successfully. The L-PDU identifier is associated with the call in order to identify the corresponding transmission.

[CANTP075] ⌈If the transmit confirmation is not received after a maximum time (equal to N_As), the CanTp module shall act as if it had received an unsuccessful transmission confirmation and any late confirmation shall be ignored. The CanTp module shall cancel (internally) the failed transmission. ⌋( )

[CANTP076] ⌈For confirmation calls, the CanTp module shall provide the function `CanTp_TxConfirmation()`.⌋( )

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.

[CANTP078] ⌈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

[CANTP079] ⌈When receiving an SF or an FF N-PDU, the CanTp module shall notify the upper layer (PDU Router) about this reception and request an Rx buffer to process the frame reassembly. The buffer request is an indication to the upper layer to reserve and lock a buffer for the incoming data. These two operations shall be performed using the `PduR_CanTpStartOfReception` function.⌋ ( )

[CANTP166] ⌈At the reception of a FF, last CF of a block or a SF, the CanTp module shall start a time-out N_Br before requesting an Rx buffer.⌋ ( )

[CANTP080] ⌈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`.

[CANTP081] ⌈After the reception of a First Frame, if the function `PduR_CanTpStartOfReception()`returns `BUFREQ_E_NOT_OK` to the CanTp module, the CanTp module shall abort the reception of this N-SDU. No Flow Control will be sent in this case.⌋ ( )

[CANTP318] ⌈After the reception of a First Frame, if the function `PduR_CanTpStartOfReception()`returns `BUFREQ_E_OVFL` to the CanTp module, the CanTp module shall send a Flow Control N-PDU with overflow status (FC(OVFLW)) and abort the N-SDU reception. If the error occurs after a Consecutive

Frame reception, the Flow Control frame shall not be sent. In case of Consecutive Frame the buffer status is returned by the PduR_CanTpCopyRxData() function. ⌋( )

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

[CANTP082] ⌈If the function PduR_CanTpStartOfReception() returns BUFREQ_E_BUSY or BUFREQ_OK with a smaller available buffer size than needed for the next block, the CanTp module shall suspend the N-SDU reception by sending the next Flow Control N-PDU with status WAIT (i.e. FC(WT)), when the N_Br timer expires. ⌋( )

[CANTP268] ⌈If the function PduR_CanTpCopyRxData() called after reception of the last Consecutive Frame of a block returns BUFREQ_E_BUSY the CanTp module shall suspend the N-SDU reception by sending the next Flow Control N-PDU with status WAIT (i.e. FC(WT)). ⌋( )

[CANTP325] ⌈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 suspend the N-SDU reception by sending the next Flow Control N-PDU with status WAIT (i.e. FC(WT)). ⌋( )

[CANTP222] ⌈Before expiration of the N_Br timer (ISO 15765-2 specification defines the following performance requirement: (N_Br+N_Ar) < 0.9*N_Bs timeout), the CanTp module shall call the service PduR_CanTpCopyRxData() with a data length 0 (zero) and NULL_PTR as data buffer during the next processing of the MainFunction. If the available buffer size is still not big enough, the CanTp module shall send a new FC(WAIT). ⌋( )

[CANTP223] ⌈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 NTFRSLT_E_WFT_OVRN occurs. ⌋( )

[CANTP311] ⌈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 NTFRSLT_E_TIMEOUT_A. ⌋( )

[CANTP224] ⌈When the Rx buffer is finally available, the CanTp module shall send a Flow Control N-PDU with ClearToSend status (FC(CTS)) and shall carry on the reception of the Consecutive Frame N-PDUs. ⌋( )

[CANTP269] ⌈After reception of each Consecutive Frame the CanTp shall call the `PduR_CanTpCopyRxData()` function with a PduInfo pointer containing data buffer and 6 or 7 data length (or less in case of the last CF). The output pointer parameter provides CanTp with available Rx buffer size after data have been copied. ⌋( )

[CANTP312] ⌈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). ⌋( )

[CANTP313] ⌈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 NTFRSLT_E_TIMEOUT_CR. ⌋( )

[CANTP270] ⌈In case that remaining buffer is too small for the next Consecutive Frame reception the CanTp might call `PduR_CanTpCopyRxData()` with a data length 0 (zero) and NULL_PTR as data buffer until the available buffer size is big enough. These calls might be performed in the time limit of STmin. ⌋( )

[CANTP271] ⌈If the `PduR_CanTpCopyRxData()` returns `BUFREQ_E_NOT_OK` or `BUFREQ_E_BUSY` after reception of a Consecutive Frame in a block the CanTp shall abort the reception of N-SDU. ⌋( )

[CANTP314] ⌈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 NTFRSLT_E_WRONG_SN. ⌋( )

[CANTP084] ⌈When the transport reception session is completed (successfully or not) the CanTp module shall call the upper layer notification service `PduR_CanTpRxIndication()`.⌋( )

[CANTP277] ⌈With regard to FF N-PDU reception, the content of the Flow Control N-PDU depends on the `PduR_CanTpStartOfReception()` service result. ⌋( )

[CANTP064] ⌈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. ⌋( )

[CANTP278] ⌈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. ⌋( )

[CANTP067] ⌈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. ⌋( )

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, the full Tx N-SDU length to be sent and the number of bytes already available in the upper layer buffer.

[CANTP225] ⌈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. ⌋( )

[CANTP226] ⌈After a transmission request from the upper layer, the CanTp module shall call `PduR_CanTpCopyTxData` at least once to request the necessary transmit buffer. ⌋( )

[CANTP167] ⌈After a transmission request from upper layer, the CanTp module shall start time-out N_Cs before the call of `PduR_CanTpCopyTxData`. If a buffer is not available before the timer elapsed, the CanTp module shall abort the communication.

The available Tx buffer can be smaller than the full Tx N-SDU data length. ⌋( )

[CANTP086] ⌈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 (address of a data buffer and up to 6 or 7 bytes as data length depending on frame type). ⌋( )

[CANTP272] ⌈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 and upper layers can return a buffer of free length. ⌋( )

If the upper layer cannot make a Tx 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), the `PduR_CanTpCopyTxData()` function returns `BUFREQ_E_NOT_OK`.

[CANTP087] ⌈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 NTFRSLT_E_NOT_OK. ⌋( )

[CANTP279] ⌈If upper layer temporarily has no Tx buffer available, the PduR_CanTpCopyTxData() function returns BUFREQ_E_BUSY. ⌋( )

[CANTP184] ⌈If the PduR_CanTpCopyTxData() function returns BUFREQ_E_BUSY, the CanTp module shall later (implementation specific) retry to receive a buffer. ⌋( )

[CANTP185] ⌈If no buffer 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. ⌋( )

[CANTP280] ⌈If buffer 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 NTFRSLT_E_NOT_OK. ⌋( )

[CANTP089] ⌈When the Tx buffer is available, the CanTp module shall resume the transmission of the N-SDU. ⌋( )

[CANTP310] ⌈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 NTFRSLT_E_TIMEOUT_A. ⌋( )

[CANTP309] ⌈If a FC frame is received with the FS set to OVFLW the CanTp module shall abort the transmit request and notify the upper layer by calling the callback function PduR_CanTpTxConfirmation() with the result NTFRSLT_E_NO_BUFFER. ⌋( )

[CANTP317] ⌈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 NTFRSLT_E_INVALID_FS. ⌋( )

[CANTP315] ⌈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). ⌋( )

[CANTP316] ⌈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 NTFRSLT_E_TIMEOUT_BS. ⌋( )

[CANTP090] ⌈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 NTFRSLT_OK.⌋()

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 indicator returned by 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.

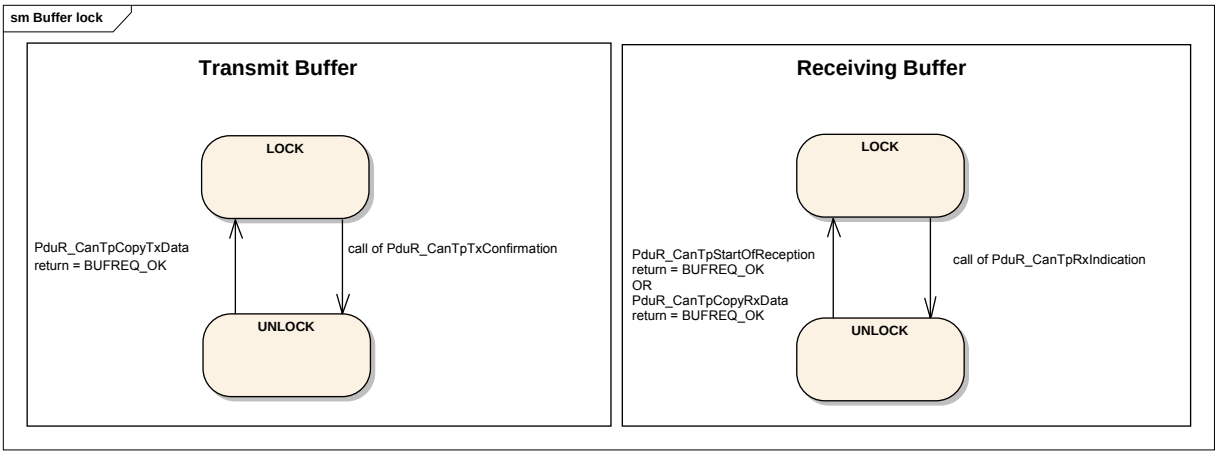


Figure 5: Tx and Rx Buffer locking

It is assumed that upper layer module has locked the buffer when it returns a status `BUFREQ_OK` to a `PduR_CanTpCopyTxData()` or `PduR_CanTpStartOfReception()` call 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.

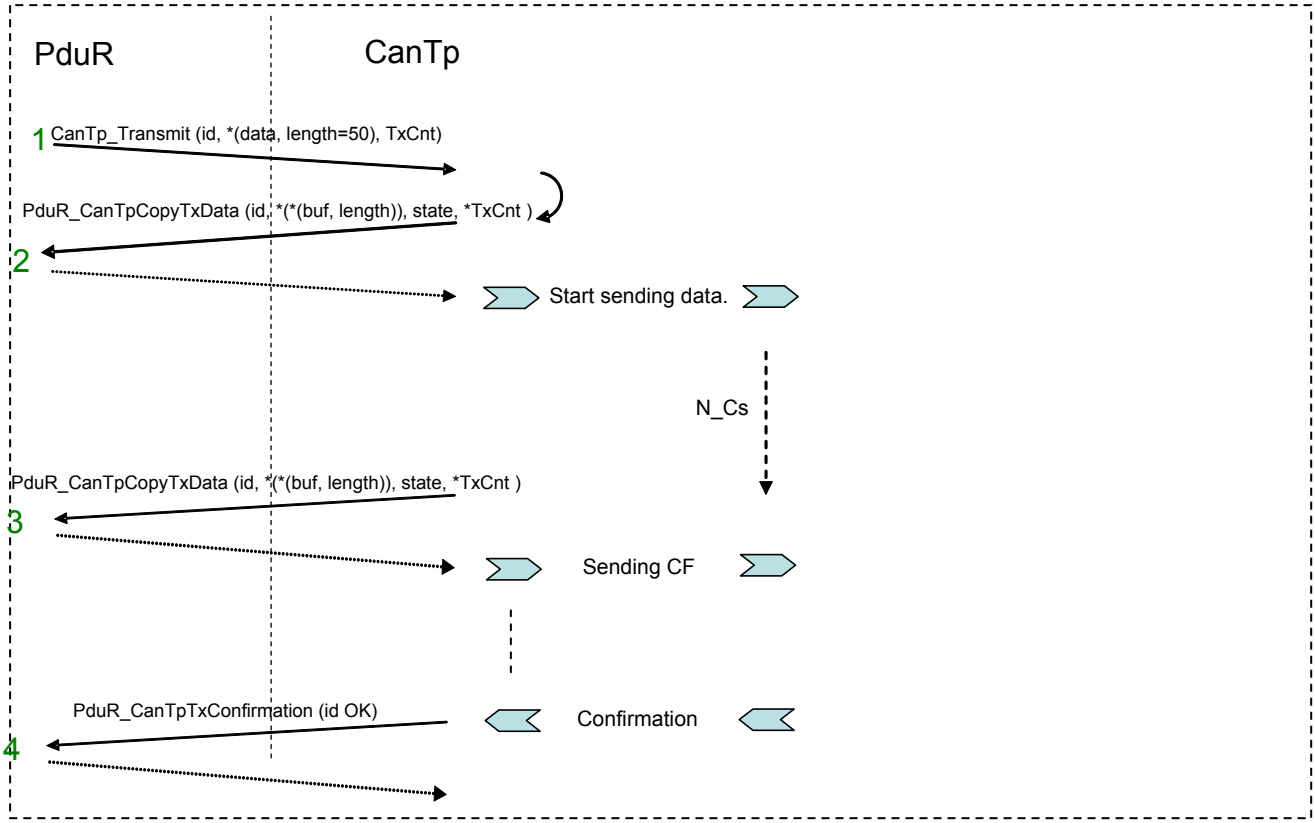
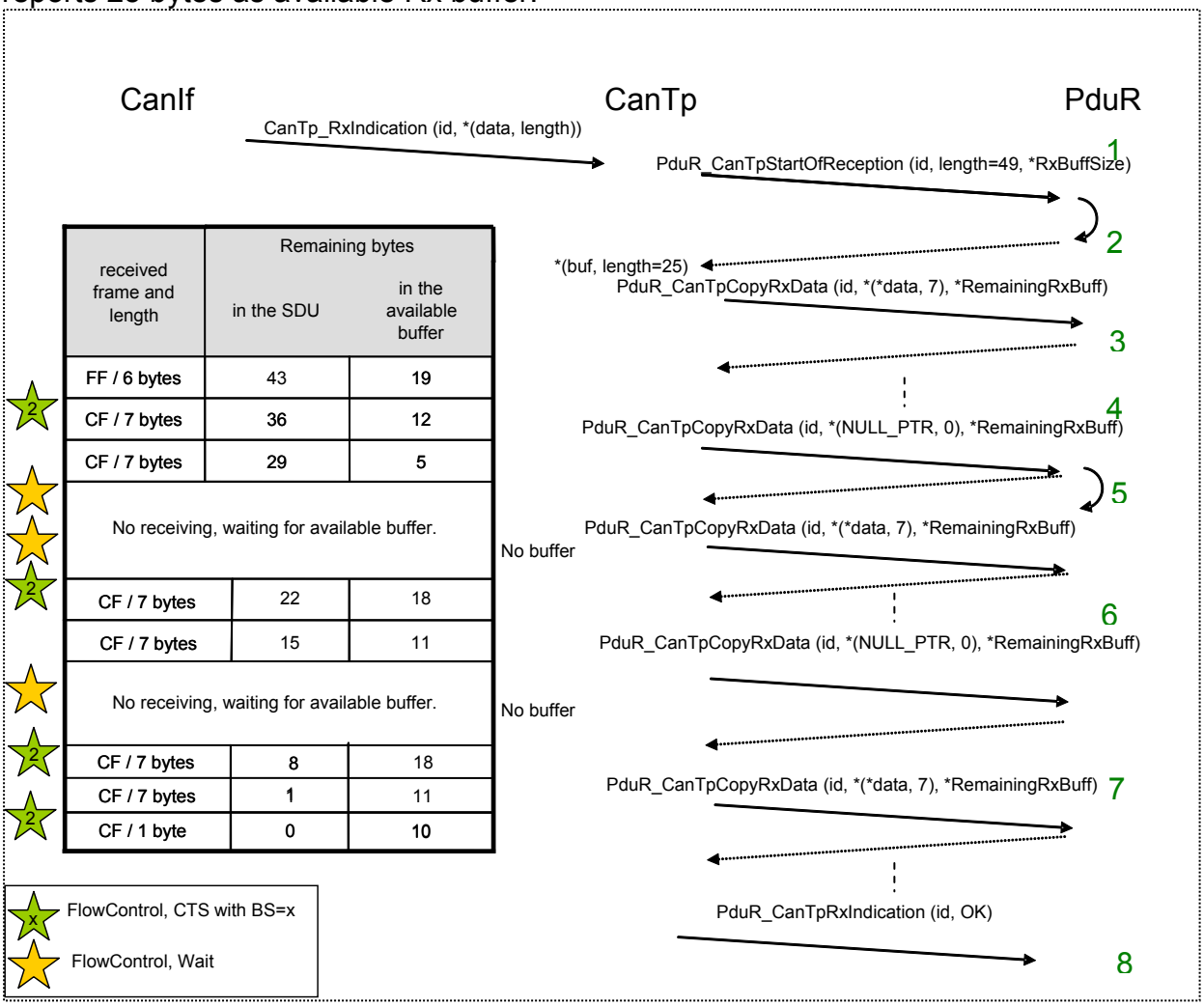


Figure 6: Example of transmit process

1: The PduR asks for the transmission of 50 data bytes;
2: The CanTp asks the PduR for the data by requesting the buffer containing 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.



received frame and length	Remaining bytes in the SDU	Remaining bytes in the available buffer
FF / 6 bytes	43	19
CF / 7 bytes	36	12
CF / 7 bytes	29	5
No receiving, waiting for available buffer.		
CF / 7 bytes	22	18
CF / 7 bytes	15	11
No receiving, waiting for available buffer.		
CF / 7 bytes	8	18
CF / 7 bytes	1	11
CF / 1 byte	0	10

Figure 7: Example of receiving process

1: The CanIf notifies a new reception with CanTp_RxIndication(). The CanTp asks the PduR for a buffer in order to store the received data;
2: The PduR returns a buffer of 25 bytes (by conception in this example, it is not able to return a buffer of 49 bytes directly);
3: The CanTp manages the payload data reception until the buffer is full (on the second consecutive frame). On this second consecutive frame, the upper layer can only store 5 bytes in the buffer. This buffer is not enough for the next block (two Consecutive Frames), so a new buffer is needed;

4: The CanTp asks the PduR for a new buffer in order to store the data received subsequently. This is done by calling PduR_CanTpCopyRxData() with 0 as data length and NULL_PTR as data. This is done till a sufficient buffer is available;
5: When the buffer is finally available, the CanTp will continue the reception of the next Consecutive Frames block;
6: After the last copy of second consecutive frame of the block, the size of the available buffer is 11 Bytes (not enough for the next block), so the CanTp needs an available buffer of at least 14 Bytes;
7: When the buffer is available the CanTp will continue the reception;
8: The CanTp informs the PduR of the end of reception by a call to PduR_CanTpRxIndication().

The CAN Transport Layer will compute the BS values (See CANTP067) depending on:
- maximum configured value for this N-SDU,
- number of free bytes inside the available buffer,
- amount of receiving bytes.

If the BS value is equal to 0 the buffer should be sized to a value equal or larger than the number of bytes to be received.

7.3.4 Protocol parameter setting services

[CANTP091] ⌈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.

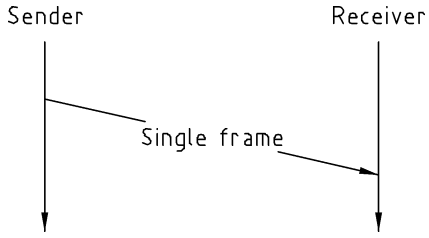


Figure 8: Example of single part message

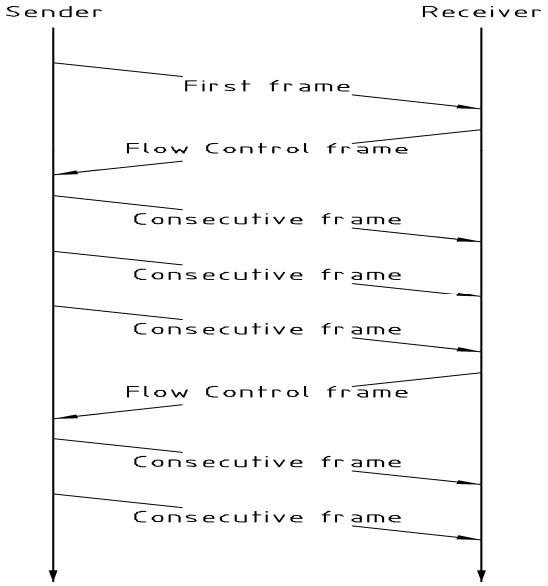


Figure 9: Example of multiple parts message

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 [13]).

[CANTP092] ⌈The CanTp module shall provide 1 to n communication (i.e. functional addressing [13]), 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.

[CANTP093] ⌈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 generate, if the development error detection is enabled, a development error CANTP_E_INVALID_TATYPE. ⌋( )

7.3.6 Relationship between CAN NSduId and CAN LSduId

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

[CANTP035] ⌈A CAN NSduId shall only be linked to one CAN LSduId that is used to transmit SF, FF, FC and CF frames. ⌋(BSW01068, BSW01069, BSW01071, BSW01078)

[CANTP281] ⌈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. ⌋( )

[CANTP282] ⌈FC protocol data units give receivers the possibility of controlling senders' data flow by authorizing or delaying transmission of subsequent CF N-PDUs. ⌋( )

[CANTP283] ⌈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. ⌋( )

[CANTP094] ⌈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. ⌋( )

[CANTP284] ⌈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. ⌋( )

[CANTP095] ⌈Therefore, in extended addressing N-PDU reception, the CanTp 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.

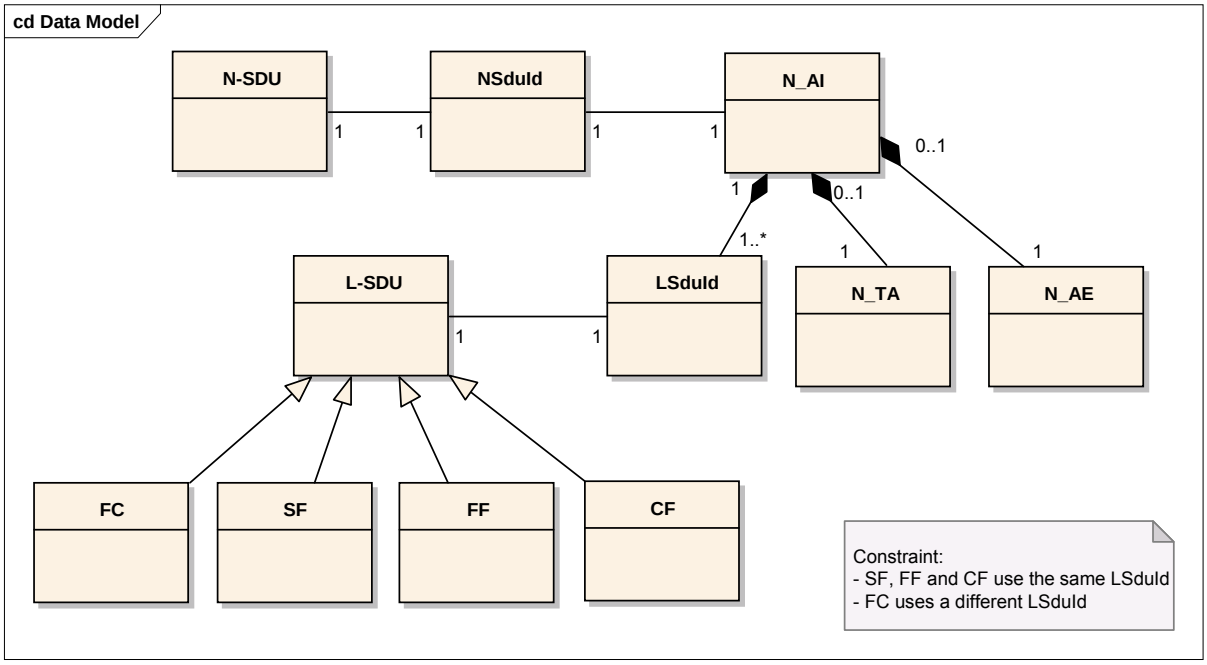


Figure 10: Possible links between NSduId and LSduId

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).

[CANTP096] ⌈The CanTp module shall support several connections simultaneously.⌋(BSW01066)

[CANTP120] ⌈It shall be possible to configure concurrent connections in the CanTp module.⌋(BSW01066)

[CANTP285] ⌈The connection channels are only destined for CAN TP internal use, so they are not accessible externally.⌋( )

[CANTP286] ⌈All the necessary information (Channel number, Timing parameter …) is configured inside the CAN Transport Layer module.⌋( )

[CANTP121] ⌈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.⌋(BSW01066)

[CANTP122] ⌈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.⌋(BSW01066)

[CANTP190] ⌈The CanTp module shall route the N-SDU through the correctly configured connection channel.⌋( )

[CANTP287] ⌈The CanTp module does not allow for the receiving or the transmission of N-SDU with the same identifier in parallel, 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.

[CANTP288] ⌈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.⌋( )

[CANTP289] ⌈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.⌋( )

[CANTP123] ⌈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. ⌋ (BSW01066)

[CANTP124] ⌈If the configured receiving connection channel is in use (state CANTP_RX_PROCESSING), on reception of new data (First Frame reception) the CanTp module shall abort the reception in progress and shall process the received frame as the start of a new reception. ⌋ (BSW01066)

[CANTP248] ⌈A Tx N-PDU Id shall not be used on two or more different connection channels. An Rx N-PDU Id can only be used on two or more different connection channels if extended addressing is used in relation with this N-PDU Id. ⌋ ( )

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.

[CANTP114] ⌈The CanTp module shall allow configuration of the padding activation at pre-compile time, by using parameter CanTpRxPaddingActivation. The configuration parameter is specific on individual received or sent N-SDU. ⌋ ( )

[CANTP040] ⌈If the CanTpTxPaddingActivation parameter is set to ON, the CanTp module shall only transfer N-PDU with a length of eight bytes (i.e. DLC = 8) between the CanTp and the CanIf, unused bytes in N-PDU shall be updated with CANTP_PADDING_BYTE. Thus, a received N-PDU shorter than 8 bytes will be considered corrupt by CanTp. ⌋ (BSW01073)

[CANTP098] ⌈If the CanTpRxPaddingActivation parameter is set to OFF, the CanTp module shall check the frame data length. If a frame is received with an unexpected datalength (check only for too short DLCs), the frame shall be ignored. ⌋ (BSW01073)

[CANTP116] ⌈In both padding and no padding modes, the CanTp module shall only transfer used data bytes to the upper layer. ⌋ (BSW01073)

[CANTP059] ⌈The value used for padding bytes is configurable via configuration parameter CANTP_PADDING_BYTE (see CanTp298_Conf). ⌋ (BSW01086)

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.

[CANTP057] ⌈If unexpected frames are received, the CanTp module shall behave according to the tables below.⌋(BSW01082, BSW01117, BSW01149)

[CANTP290] ⌈Those tables consider the actual CanTp internal status (CanTp status). Table 1 specifies the behavior on the half duplex implementation while table 2 defines the behavior for full duplex channels.⌋(BSW01117, BSW01149)

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	Ignore	Ignore	Ignore	If awaited, process the FC N-PDU, otherwise ignore it.	Ignore
Segmented Receive in progress	Terminate the current reception, report an indication, with parameter Result set to NTFRSLT_E_UNEXP_PDU, 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 NTFRSLT_E_UNEXP_PDU, 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)	Ignore	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

Table 1: Handling of unexpected N-PDU arrivals for half duplex channels

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

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
Segmented Receive in progress	Terminate the current reception, report an indication, with parameter Result set to NTFRSLT_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 NTFRSLT_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[2]	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

Table 2: Handling of N-PDU arrivals for full duplex channels

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.

The general requirements document of AUTOSAR [3] specifies that all basic software modules must distinguish (according to the product life cycle) two error types:

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

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

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

[CANTP008] ⌈On errors and exceptions, the CanTp module shall not modify its current module state (see Figure 4: CAN Transport Layer life cycle) but shall simply report the error event.⌋(BSW00339)

[CANTP291] ⌈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).⌋( )

[CANTP101] ⌈Development error values are of type uint8.⌋(BSW385, BSW386, BSW00337, BSW00327, BSW00331)

[CANTP293] ⌈The CanTp module shall be able to detect the following errors and exceptions depending on its configuration (development/production):⌋( )

Type or error	***Relevance***	***Related error code***	***Value [hex]***
API service called with wrong parameter(s): When CanTp_Transmit is called for a none configured PDU identifier or with an identifier for a received PDU.	Development	Could be a combination of: CANTP_E_PARAM_CONFIG CANTP_E_PARAM_ID	 0x01 0x02
API service called with a NULL pointer. In case of this error, the API service shall return immediately without any further action, besides reporting this development error.	Development	CANTP_E_PARAM_POINTER	0x03
API service used without module initialization : On any API call except CanTp_Init() and CanTp_GetVersionInfo() if CanTp is in state CANTP_OFF"	Development	CANTP_E_UNINIT	0x20
Invalid Transmit PDU identifier (e.g. a service is called with an inexistent Tx PDU identifier)	Development	CANTP_E_INVALID_TX_ID	0x30
Invalid Receive PDU identifier (e.g. a service is called with an inexistent Rx PDU identifier)	Development	CANTP_E_INVALID_RX_ID	0x40
Invalid Transmit buffer address	Development	CANTP_E_INVALID_TX_BUFFER	0x50

(e.g. the Tx buffer address is inaccessible or NULL)			
Invalid Receive buffer address (e.g. the Rx buffer address is inaccessible or NULL)	Development	CANTP_E_INVALID_RX_BUFFER	0x60
Invalid data length of the transmit PDU (e.g. a transmit N-SDU has a length equal to zero)	Development	CANTP_E_INVALID_TX_LENGTH	0x70
Invalid data length of the receive PDU (e.g. a receive FF N-PDU has a FF_DL equal to zero)	Development	CANTP_E_INVALID_RX_LENGTH	0x80
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	Development	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	Development	CANTP_E_OPER_NOT_SUPPORTED	0xA0
Another error occurred during a reception or a transmission: any protocol timeout error or implementation specific error	Development	CANTP_E_COM	0xB0
Event reported on completion of a reception operation	Development	CANTP_E_RX_COM	0xC0
Event reported on completion of a transmission operation	Development	CANTP_E_TX_COM	0xD0

7.5 Error detection

[CANTP006] ⌈The detection of development errors is configurable (*ON* / *OFF*) at pre-compile time.
The switch `CanTpDevErrorDetect` (see chapter 10) should activate or deactivate the detection of all development errors.⌋(BSW00350)

[CANTP132] ⌈If the `CanTpDevErrorDetect` switch is enabled API parameter checking is enabled. The detailed description of the detected errors can be found in chapter 7.4 and chapter 8.⌋(BSW00323)

[CANTP133] ⌈The detection of production code errors cannot be switched off.⌋( )

[CANTP161] ⌈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. ⌋(BSW406)

7.6 Error notification

[CANTP134] ⌈Detected development errors will be reported to the error hook of the Development Error Tracer (DET) if the pre-processor switch *CanTpDevErrorDetect* is set. ⌋( )

[CANTP193] ⌈Production errors shall be reported to Diagnostic Event Manager. ⌋( )

The Development Error Tracer module is merely an aid to BSW development and integration. The API is defined, but the functionality can be chosen and implemented according to the development needs (e.g. error count, send error information via a serial interface to an external logger, and so on).

[CANTP021] ⌈The CanTp module shall use the Development Error Tracer service [8]:
`Std_ReturnType Det_ReportError(ModuleId, InstanceId, ApiId, ErrorId)` to report development errors. ⌋(BSW00369)

[CANTP115] ⌈The header file of the CanTp module, `CanTp.h`, shall provide a module ID, called `CANTP_MODULE_ID`, sets to the value 0x23. ⌋( )

The following figure describes how this function can be used when the Development Error Tracer is on.

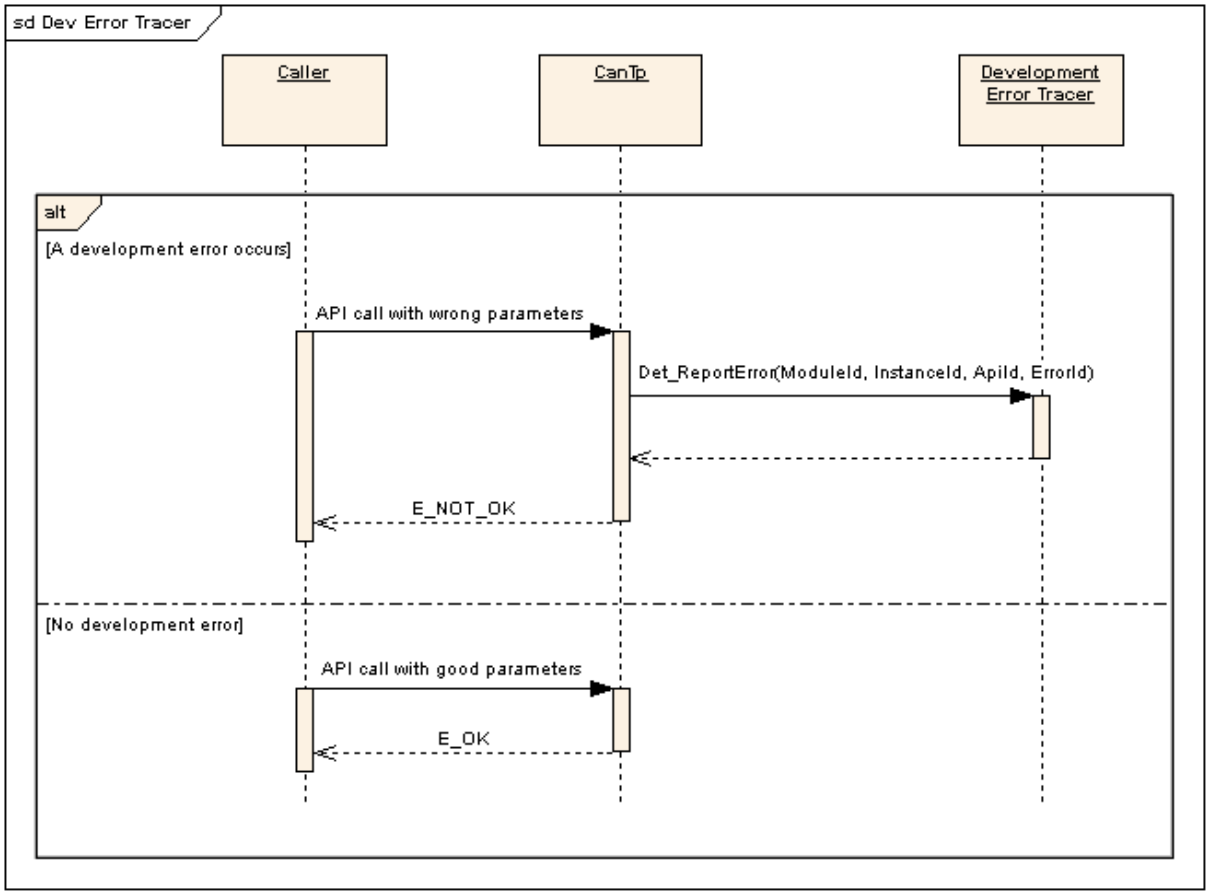


Figure 11: Development error reporting

[CANTP294] ⌈As shown in the above figure, when a development error occurs the CanTp returns the value E_NOT_OK. The error description is only reported via the API of the Development Error Tracer module. ⌋ ( )

[CANTP229] ⌈If the task was aborted due to As, Bs, Cs, Ar, Br, Cr timeout, the CanTp module shall raise the DET error CANTP_E_RX_COM (in case of a reception operation) or 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 DET error CANTP_E_COM. ⌋ ( )

7.7 AUTOSAR debugging concept

The following requirements offer support for implementation of AUTOSAR debugging concept.

[CANTP249] ⌈Each variable that shall be accessible by AUTOSAR Debugging, shall be defined as global variable. ⌋ ( )

[CANTP250] ⌈All type definitions of variables that shall be debugged shall be accessible by the header file CanTp.h. ⌋( )

[CANTP251] ⌈The declaration of variables in the header file shall be such, that it is possible to calculate the size of the variables by C-"sizeof".⌋( )

[CANTP252] ⌈Variables available for debugging shall be described in the CanTp Software Module Description. ⌋( )

8 API specification

8.1 Imported types

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

[CANTP209]

Module	Imported Type
ComStack_Types	BufReq_ReturnType
	NotifResultType
	PduIdType
	PduInfoType
	PduLengthType
	RetryInfoType
	TPParameterType
Std_Types	Std_ReturnType
	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.

[CANTP002] ⌈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. ⌋ (BSW00353)

[CANTP296] ⌈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

Name:	`CanTp_ConfigType`
Type:	`Structure`
Range:	Implementation specific.
Description:	Data structure type for the post-build configuration parameters.

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

[CANTP003] ⌈The following provides the API Naming convention for the CanTp services:

- The service name format is CanTp_<ServiceName>(…)
- <ServiceName>: is the name of the service primitive with first letter of each word upper case and consecutive letters lower case⌋ (BSW00310)

8.3.1 CanTp_Init

[CANTP208]

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.

⌋ (BSW101, BSW00358, BSW00414)

After power up, CanTp is in a state called CANTP_OFF (see CANTP168). 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 CANTP170 and CANTP030).

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.

[CANTP199] ⌈The CanTp module's environment shall call CanTp_Init before using the CanTp module for further processing.⌋ ( )

[CANTP320] ⌈If DET is enabled the function CanTp_Init shall rise CANTP_E_PARAM_POINTER error if the argument is a NULL pointer and return without any action. ⌋( )

8.3.2 CanTp_ GetVersionInfo

[CANTP210]

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.

⌋( )

[CANTP162] ⌈The function CanTp_GetVersionInfo shall return the version information of this module. The version information includes:

- Module Id
- Vendor Id
- Vendor specific version numbers (BSW00407). ⌋(BSW407)

[CANTP163] ⌈The function CanTp_GetVersionInfo shall be pre compile time configurable (`On/Off`) by the configuration parameter: `CANTP_VERSION_INFO_API.` ⌋(BSW407)

[CANTP297] ⌈CANTP_VERSION_INFO_API shall be used to switch CanTp_GetVersionInfo API On or OFF. ⌋( )

[CANTP218] ⌈If source code for caller and callee of CanTp_GetVersionInfo is available, the CanTp module should realize CanTp_GetVersionInfo as a macro, defined in the module's header file. ⌋( )

[CANTP319] ⌈If DET is enabled the function CanTp_GetVersionInfo shall rise CANTP_E_PARAM_POINTER error if the argument is a NULL pointer and return without any action. ⌋( )

Note that the function CanTp_GetVersionInfo can be called before initialization of the CanTp module.

8.3.3 CanTp_Shutdown

[CANTP211]

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.

⌋( )

[CANTP202] ⌈The function CanTp_Shutdown shall close all pending transport protocol connections, free all resources and set the CanTp module into the CANTP_OFF state. ⌋( )

[CANTP200] ⌈The function CanTp_Shutdown shall not raise a notification about the pending frame transmission or reception. ⌋( )

8.3.4 CanTp_Transmit

[CANTP212]

Service name:	CanTp_Transmit	

Syntax:	`Std_ReturnType CanTp_Transmit(`
	`  PduIdType CanTpTxSduId,`
	`  const PduInfoType* CanTpTxInfoPtr`
	`)`	

Service ID[hex]:	0x03	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant	
Parameters (in):	CanTpTxSduId	This parameter contains the unique CanTp module identifier of the CAN N-SDU to be transmitted. Range: 0..(maximum number of L-PDU IDs received) - 1
	CanTpTxInfoPtr	An indicator of a structure with CAN N-SDU related data: indicator of a CAN N-SDU buffer and the length of this buffer.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The request can be started successfully E_NOT_OK: The request cannot be started (e.g. a transmit request is in progress with the same N-SDU identifier)
Description:	This service is used to request the transfer of segmented data.	

⌋( )

[CANTP231] ⌈If data length is less than 7 or 6 (depending on normal or extended addressing format), the function CanTp_Transmit shall send a SF N-PDU. ⌋( )

[CANTP232] ⌈If data length is greater than 7 or 6 (depending on normal or extended addressing format), the function CanTp_Transmit shall initiate a multiple frame transmission session. ⌋( )

[CANTP204] ⌈The CanTp module shall notify the upper layer by calling the `PduR_CanTpTxConfirmation` callback when the transmit request has been completed. ⌋( )

[CANTP205] ⌈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).

[CANTP206] ⌈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. ⌋( )

[CANTP298] ⌈CanTp has limited buffering capability, and hence the N-SDU payload to be transmitted is not copied internally. The CAN Transport Layer works on the memory area referenced by the CAN N-SDU pointer obtained within 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.

[CANTP299] ⌈When the upper layer calls this function, only the data length information of the structure indicated by CanTpTxInfoPtr has to be used. Its value indicates the payload length of the N-SDU, which is to be transmitted.
To access a Tx buffer, the CAN Transport Layer should call the PduR_CanTpCopyTxData service. ⌋( )

[CANTP321] ⌈If DET is enabled the function CanTp_Transmit shall rise CANTP_E_PARAM_POINTER error if the argument *CanTpTxInfoPtr* is a NULL pointer and return without any action. ⌋( )

8.3.5 CanTp_CancelTransmit

[CANTP246]

Service name:	CanTp_CancelTransmit	

Syntax:	`Std_ReturnType CanTp_CancelTransmit(`
	`  PduIdType CanTpTxSduId`
	`)`	

Service ID[hex]:	0x08	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	CanTpTxSduId	This parameter contains the unique CanTp module identifier of the N-SDU to be cancelled for transmission. Range: 0..(maximum number of L-PDU IDs received) - 1
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Cancellation request of the specified N-SDU is accepted. E_NOT_OK: Cancellation request is rejected; the reason can be that request is issued for an N-SDU that is not segmented, request is issued after the last CF has been requested for transmission or cancellation is not possible for the related N-SDU due to configuration.
Description:	This service primitive is used to cancel the transfer of a pending CAN N-SDU. When the function returns, no transmission is in progress anymore with the given N-SDU identifier.	

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

[CANTP254] ⌈If development error detection is enabled the function CanTp_CancelTransmit shall check the validity of CanTpTxSduId parameter. If the parameter value is invalid, the CanTp_CancelTransmit function shall raise the development error CANTP_E_PARAM_ID and return E_NOT_OK.
If the parameter value indicates a cancel transmission request for an N-SDU that it is not on transmission process the CanTp module shall raise the DET error CANTP_E_OPER_NOT_SUPPORTED and the service shall return E_NOT_OK. ⌋( )

[CANTP256] ⌈The CanTp shall abort the transmission of the current N-SDU if the service returns E_OK. ⌋( )

[CANTP255] ⌈If the CanTp_CancelTransmit service has been successfully executed the CanTp shall call the PduR_CanTpTxConfirmation with notification result NTFRSLT_E_CANCELATION_OK. ⌋( )

8.3.6 CanTp_CancelReceive

[CANTP257]

Service name:	CanTp_CancelReceive

Syntax:	`Std_ReturnType CanTp_CancelReceive(`
	`  PduIdType CanTpRxSduId`
	`)`

Service ID[hex]:	0x09	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	CanTpRxSduId	Identifier of the received N-SDU.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Cancellation request of the specified N-SDU is accepted. E_NOT_OK: Cancellation request is rejected; the reason can be that request is issued for an N-SDU that is not segmented or request is issued for an N-SDU that is not in the reception process.
Description:	This service is used to cancel the reception of an ongoing N-SDU.	

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. ⌋( )

[CANTP260] ⌈If development error detection is enabled the function CanTp_CancelReceive shall check the validity of CanTpRxSduId parameter. If the parameter value is invalid, the CanTp_CancelReceive function shall raise the development error CANTP_E_PARAM_ID and return E_NOT_OK.
If the parameter value indicates a cancel reception request for an N-SDU that it is not on reception process the CanTp module shall raise the DET error CANTP_E_OPER_NOT_SUPPORTED and the service shall return E_NOT_OK. ⌋( )

[CANTP261] ⌈The CanTp shall abort the reception of the current N-SDU if the service returns E_OK. ⌋( )

[CANTP262] ⌈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. ⌋( )

[CANTP263] ⌈If the CanTp_CancelReceive service has been successfully executed the CanTp shall call the PduR_CanTpRxIndication with notification result NTFRSLT_E_CANCELATION_OK. ⌋( )

8.3.7 CanTp_ChangeParameter

[CANTP302]

Service name:	CanTp_ChangeParameter

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

Service ID[hex]:	0x0a	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	id	Identifier of the received N-SDU on which the reception parameter has to be changed.
	parameter	Specify the parameter to which the value has to be changed (BS or STmin).
	value	The new value of the parameter.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: request is accepted. E_NOT_OK: request is not accepted.
Description:	This service is used to request the change of reception parameters BS and STmin for a specified N-SDU.	

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). ⌋( )

[CANTP303] ⌈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. ⌋( )

[CANTP304] ⌈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. After the request for parameter change has rejected the CanTp module shall call the PduR_CanTpChangeParameterConfirmation service with notification result NTFRSLT_E_RX_ON. ⌋( )

[CANTP305] ⌈If development error detection is enabled the function CanTp_ChangeParameter shall check the validity of function parameters (Identifier, Parameter and requested value). If any of the parameter value is invalid, the CanTp_ChangeParameter function shall raise the development error CANTP_E_PARAM_ID and return E_NOT_OK. After the request for parameter change has rejected the CanTp module shall call the PduR_CanTpChangeParameterConfirmation service with notification result NTFRSLT_E_PARAMETER_NOT_OK, in case of a wrong Identifier or Parameter type value, and with notification result NTFRSLT_E_VALUE_NOT_OK in case of a wrong Value requested. ⌋( )

[CANTP306] ⌈After CanTp_ChangeParameter service has been successfully executed the (i.e. service returned E_OK) the CanTp shall call the PduR_CanTpChangeParameterConfirmation service with notification result NTFRSLT_PARAMETER_OK. ⌋( )

8.3.8 CanTp_ReadParameter

[CANTP323]

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.	

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

[CANTP324] ⌈If development error detection is enabled the function CanTp_ReadParameter shall check the validity of function parameters (Id and Parameter). If any of the parameter value is invalid, the CanTp_ReadParameter function shall raise the development error CANTP_E_PARAM_ID and return E_NOT_OK. ⌋( )

8.3.9 Main Function

[CANTP213]

Service name:	CanTp_MainFunction
Syntax:	`void CanTp_MainFunction( void )`
Service ID[hex]:	0x06
Timing:	FIXED_CYCLIC
Description:	The main function for scheduling the CAN TP.

⌋( )

[CANTP164] ⌈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. ⌋(BSW00424, BSW00373, BSW00376)

[CANTP300] ⌈The function CanTp_MainFunction is affected by configuration parameter CanTpMainFunctionPeriod. ⌋( )

8.4 Call-back notifications

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

[CANTP233] ⌈The CanTp module shall provide the function prototypes of the callback functions in the file `CanTp_Cbk.h`⌋( )

8.4.1 CanTp_RxIndication

[CANTP214]

Service name:	CanTp_RxIndication	
Syntax:	`void CanTp_RxIndication( PduIdType RxPduId, 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 I-PDU.
	PduInfoPtr	Contains the length (SduLength) of the received I-PDU and a pointer to a buffer (SduDataPtr) containing the I-PDU.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Indication of a received I-PDU from a lower layer communication module.	

⌋( )
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.

[CANTP235] ⌈The function CanTp_RxIndication shall be callable in interrupt context (it could be called from the CAN receive interrupt). ⌋( )

[CANTP322] ⌈If DET is enabled the function CanTp_RxIndication shall rise CANTP_E_PARAM_POINTER error if the argument PduInfoPtr is a NULL pointer and return without any action. ⌋ ( )

8.4.2 CanTp_TxConfirmation

[CANTP215]

Service name:	CanTp_TxConfirmation	

Syntax:	`void CanTp_TxConfirmation(`
	`  PduIdType TxPduId`
	`)`	

Service ID[hex]:	0x40	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant for different PduIds. Non reentrant for the same PduId.	
Parameters (in):	TxPduId	ID of the I-PDU that has been transmitted.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	The lower layer communication module confirms the transmission of an I-PDU.	

⌋ ( )
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.

[CANTP236] ⌈The function CanTp_TxConfirmation shall be callable in interrupt context (it could be called from the CAN transmit interrupt). ⌋ ( )

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.

[CANTP216]

API function	***Description***
CanIf_Transmit	This service initiates a request for transmission of the CAN L-PDU specified by the CanTxPduId and CAN related data in the L-PDU structure.
PduR_CanTpCopyRxData	This function is called when a transport protocol module has data to copy for the receiving module. Several calls may be made during one transportation of an I-PDU. The service shall provide the currently available buffer size when invoked with info.SduLength equal to 0.
PduR_CanTpCopyTxData	This function is called by the transport protocol module to query the transmit data of an I-PDU segment.

	Each call to this function copies the next part of the transmit data until TpDataState indicates TP_DATARETRY. In this case the API restarts to copy the data beginning at the location indicated by TpTxDataCnt. The service shall provide the size of the remaining data when invoked with info.SduLength equal to 0.
PduR_CanTpRxIndication	Called by the transport protocol module after an I-PDU has been received successfully or when an error occurred. It is also used to confirm cancellation of an I-PDU.
PduR_CanTpStartOfReception	This function will be called by the transport protocol module at the start of receiving an I-PDU. The I-PDU 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 TpSduLength equal to 0.
PduR_CanTpTxConfirmation	This function is called by a transport protocol module after the I-PDU has been transmitted on its network, the result will reveal if 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.

[CANTP217]

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

⌋ ( )

9 Sequence diagrams

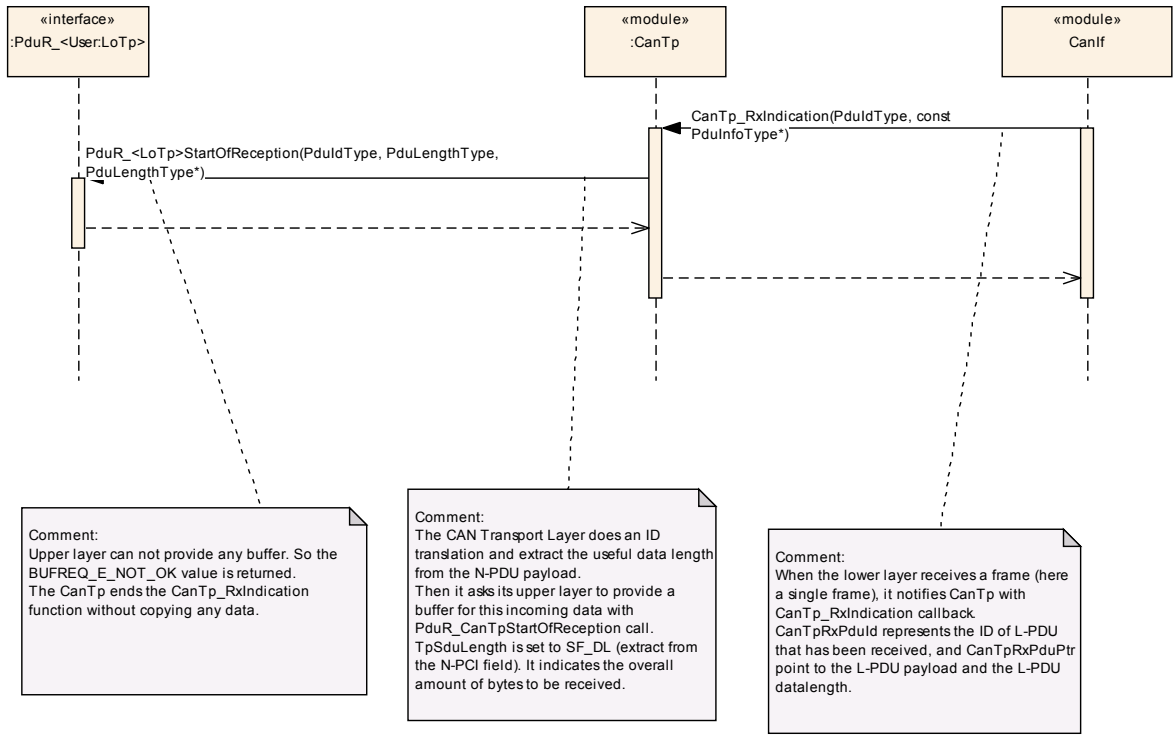
The goal of this chapter is to make it easier to understand the CAN Transport Layer by describing most of the more frequent and complicated use cases. Thus, the following diagram sequences are not exhaustive and do not reflect all the specified API possibilities.

9.1 SF N-SDU received and no buffer provided

9.1.1 Assumptions

- All input parameters are OK;
- The N-SDU data length is smaller than or equal to 7 bytes (6 bytes in the case of extended or mixed addressing format);
- Upper layer can not provide an Rx buffer.

9.1.2 Sequence diagram



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

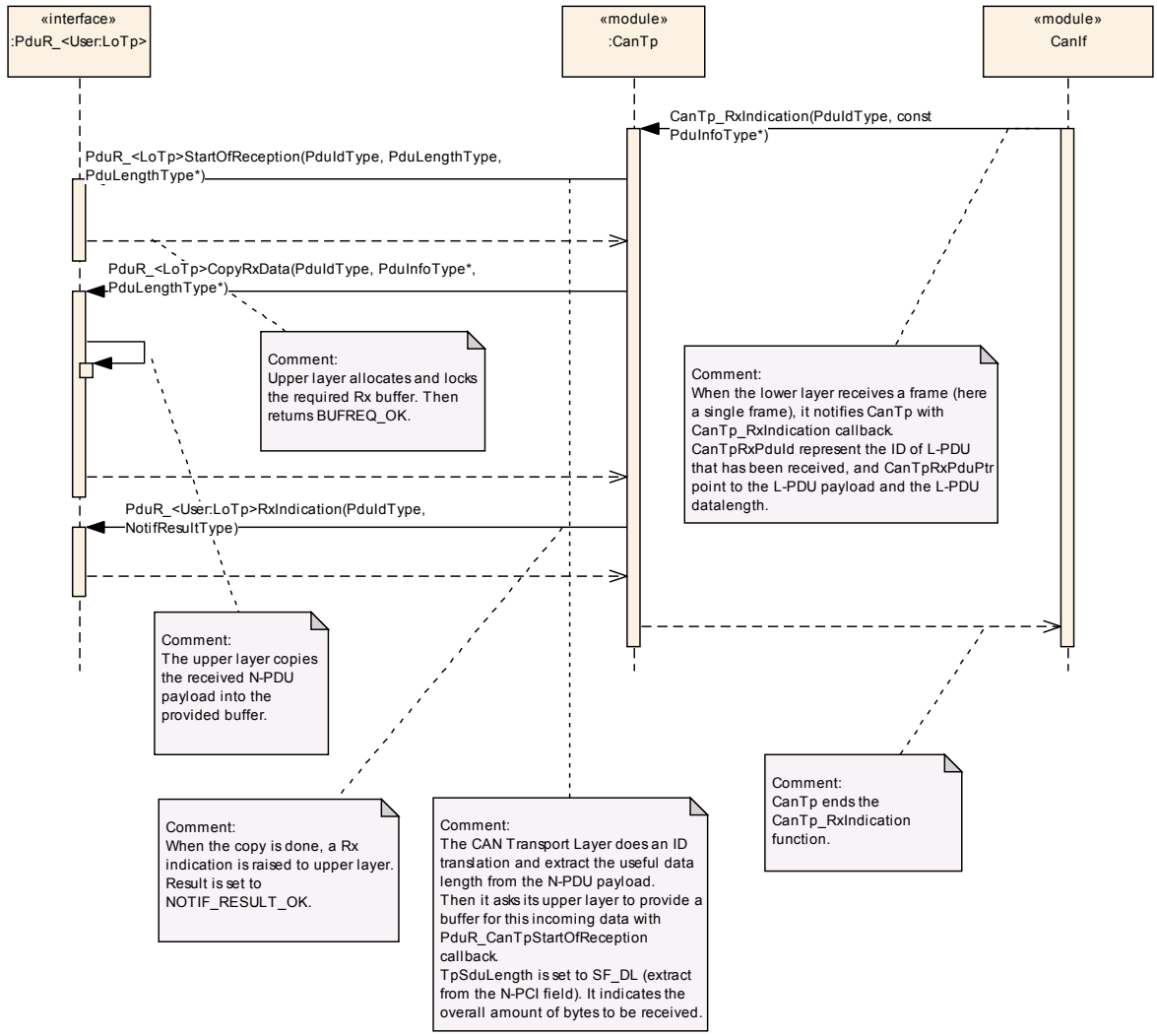
Transition	Name	Description
1	`CanTp_RxIndication ( CanTpRxPduId, CanTpRxPduPtr )`	When the lower layer receives a frame (here a single frame), it notifies CanTp by means of a CanTp_RxIndication callback. CanTpRxPduId represents the ID of L-PDU that has been received, and CanTpRxPduPtr indicates the L-PDU payload and the L-PDU data length.
2	`PduR_CanTpStartOfReception( CanTpRxSduId, TpSduLength, PduInfoPtr )`	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 provide a buffer 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 provide any buffer, so the `BUFREQ_E_NOT_OK` value is returned. The CanTp ends the CanTp_RxIndication 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 is smaller than or equal to 7 bytes (6 bytes in the case of extended addressing format);
- The SF N-PDU is successfully received.

9.2.2 Sequence diagram



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

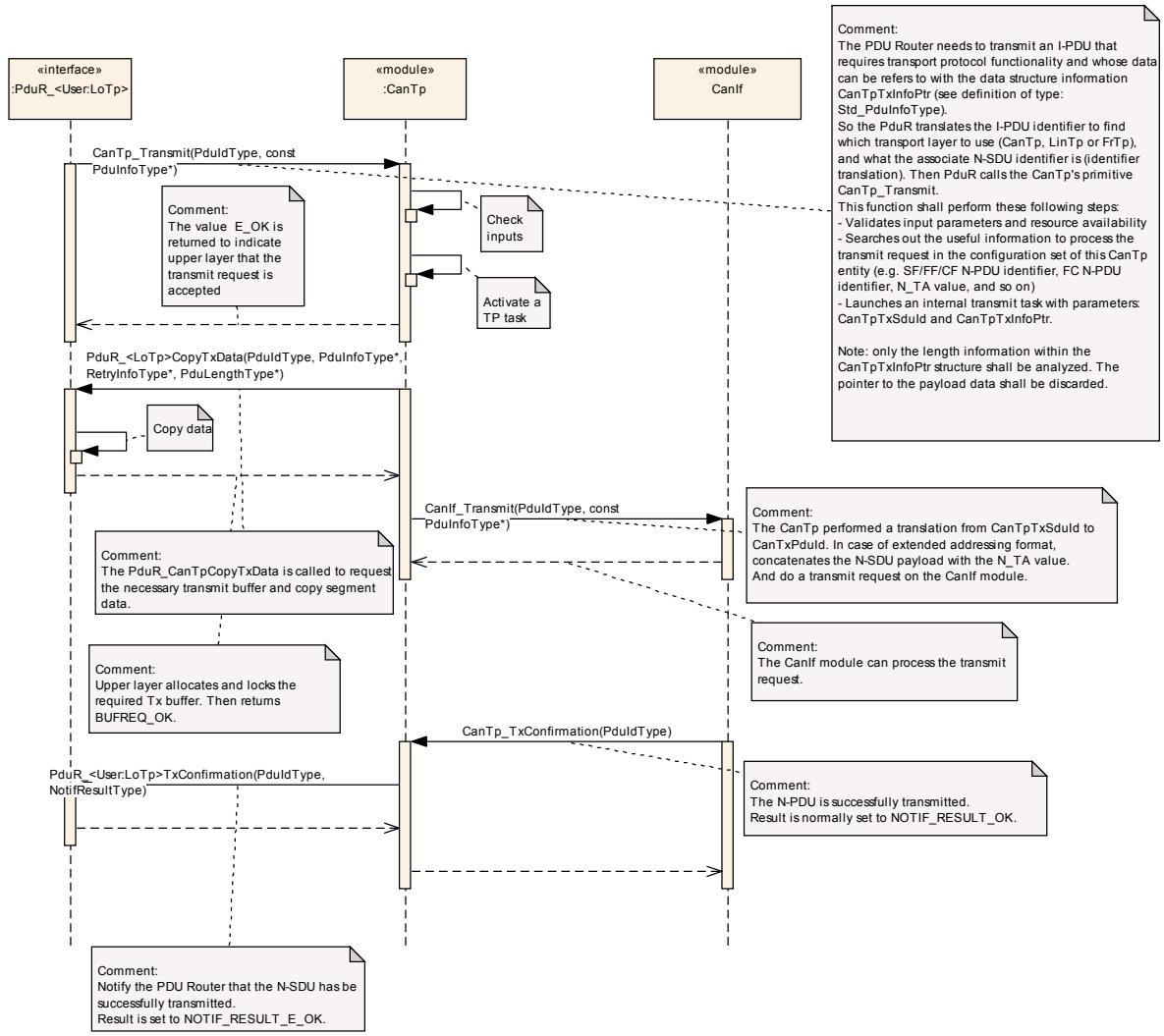
Transition	Name	Description
1	`CanTp_RxIndication ( CanTpRxPduId, CanTpRxPduPtr )`	When the lower layer receives a frame (here a single frame), it notifies CanTp by means of a CanTp_RxIndication callback. CanTpRxPduId represents the ID of the L-PDU that has been received, and CanTpRxPduPtr indicates the L-PDU payload and the L-PDU data length.
2	`PduR_CanTpStartOfReception( CanTpRxSduId, TpSduLength, PduInfoPtr )`	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 provide a buffer 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_E_OK`.
4	`PduR_CanTpCopyRxData( PduIdType, PduInfoType*, PduLengthType )`	The upper layer copies the received N-PDU payload into the buffer provided.
5	`PduR_CanTpRxIndication ( CanTpRxSduId, Result )`	When the copy is complete, an Rx indication is sent to the upper layer. The result is set to `NTFRSLT_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 is smaller than or equal to 7 bytes (6 bytes in case of extended addressing format);
- The transmission is successfully processed.

9.3.2 Sequence diagram



9.3.3 Transition description

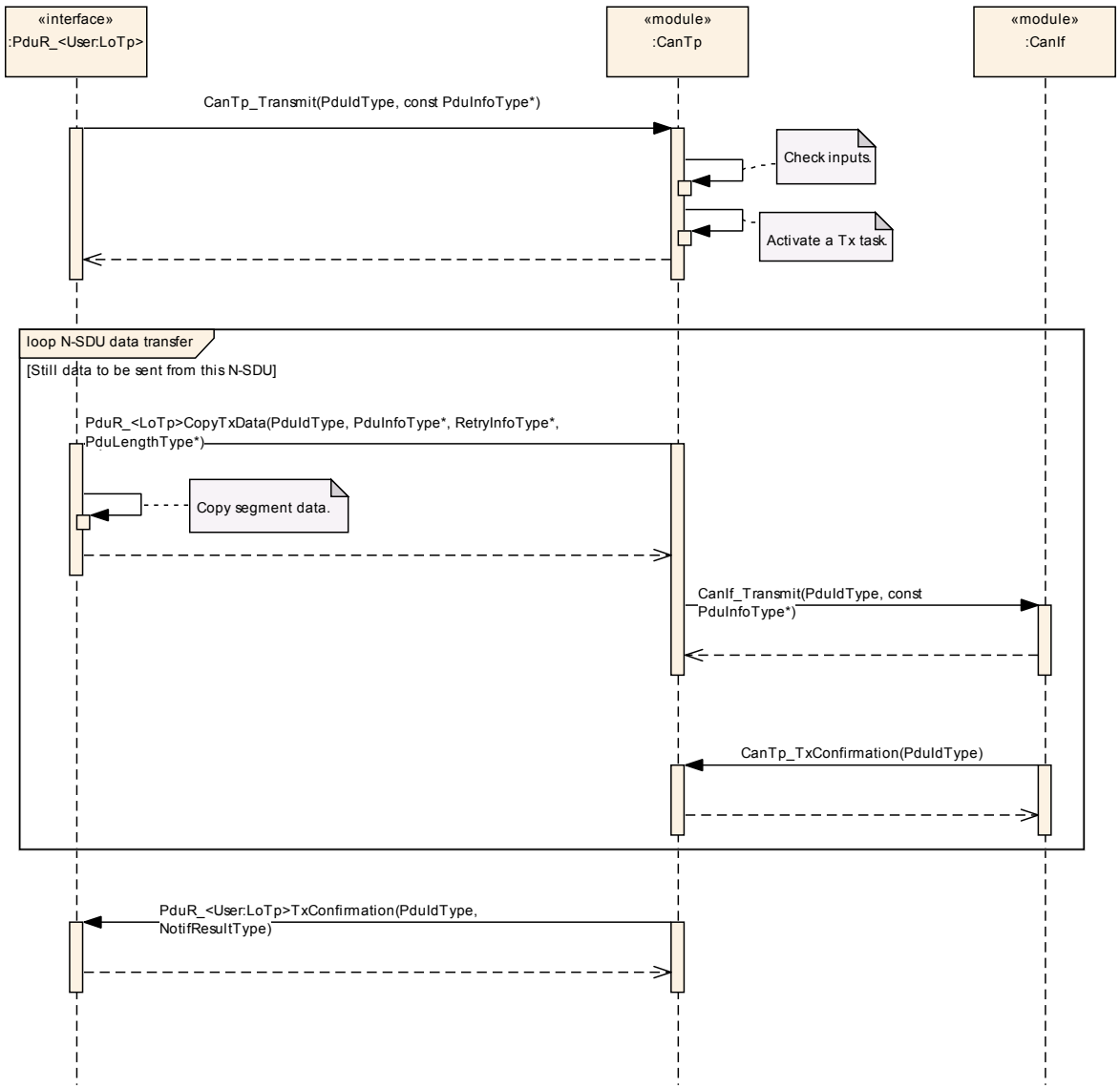
Transition	Name	Description
1	`CanTp_Transmit( CanTpTxSduId, CanTpTxInfoPtr )`	The PDU Router needs to transmit an I-PDU that requires transport protocol functionality and whose data can be refers to with the data structure information CanTpTxInfoPtr (see definition of type: Std_PduInfoType). So the 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. 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: CanTpTxSduId and CanTpTxInfoPtr. Note: only the length information within the CanTpTxInfoPtr 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
3	`PduR_CanTpCopyTxdata ( SduId, PduInfoPtr, TxState, TxCntPtr )`	The PduR_CanTpCopyTxData is called to request the necessary transmit buffer and copy segment data.
4	`BUFREQ_OK`	Upper layer copy data and locks the required Tx buffer, then returns BUFREQ_E_OK.
5	`CanIf_Transmit( CanTxPduId, PduInfoPtr )`	The CanTp performs a translation from CanTpTxSduId to CanTxPduId. In case of extended addressing format, it concatenates the N-SDU payload with the N_TA value, to perform a transmit request on the CanIf module.
6	`E_OK`	The CanIf module can process the transmit request.
7	`CanTp_TxConfirmation( CanTpTxPduId, )`	The N-PDU is successfully transmitted.
8	`PduR_CanTpTxConfirmation ( CanTpTxSduId, Result )`	Notifies the PDU Router that the N-SDU has been successfully transmitted. Consequently, the PduInfoType structure has to be unlocked. Result is set to NTFRSLT_OK.

9.4 Transmit request of larger N-SDU

9.4.1 Assumptions

- All input parameters are OK;
- The N-SDU data length is larger than 7 bytes (6 bytes in case of extended or mixed addressing format);
- The transmission is successfully processed.

9.4.2 Sequence diagram



9.4.3 Transition description

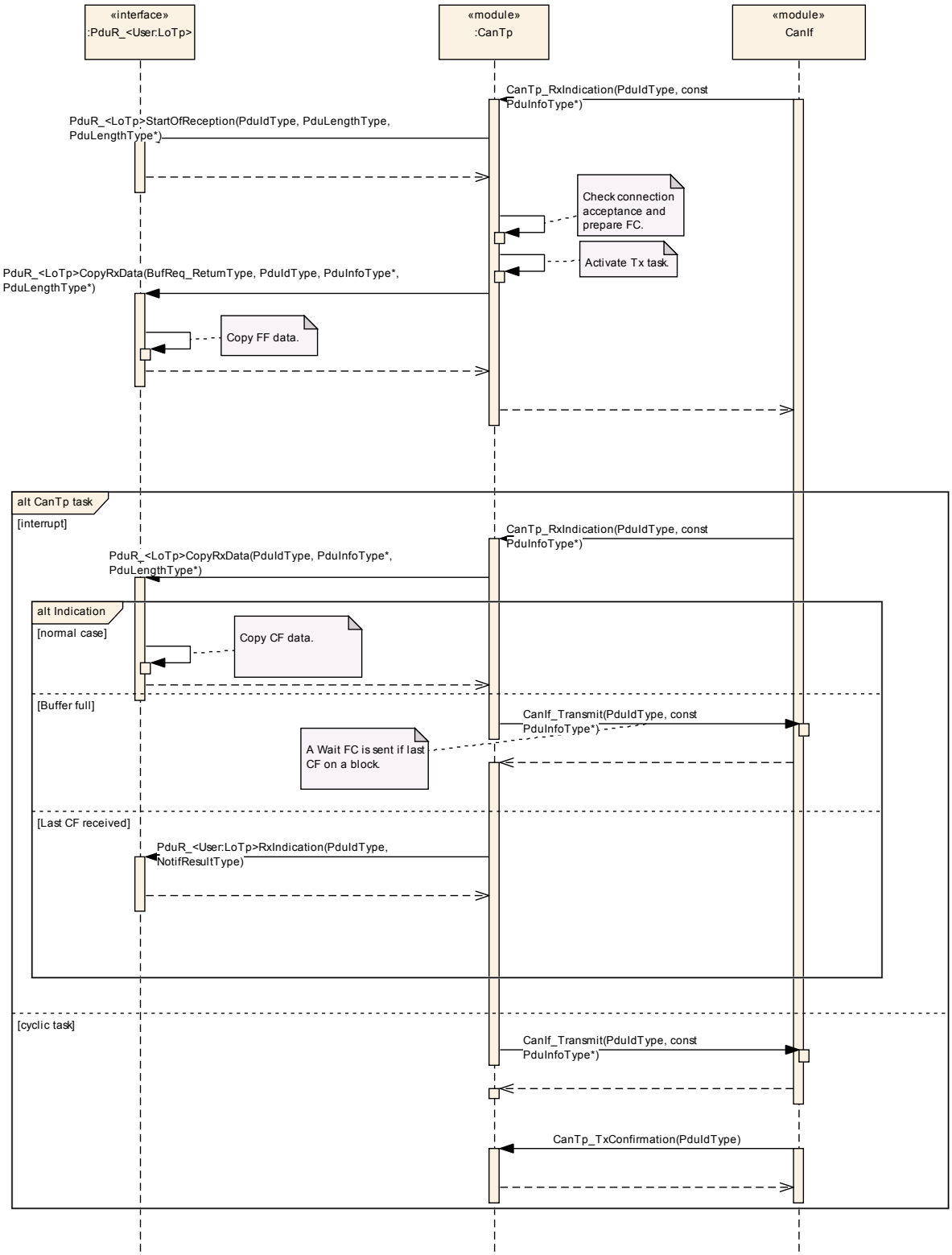
Transition	Name	Description
1	`CanTp_Transmit ( CanTpTxSduId, CanTpTxInfoPtr )`	The PDU Router needs to transmit an I-PDU that requires transport protocol functionality and whose data refers to with the data structure information CanTpTxInfoPtr (see definition of type: PduInfoType). So the PduR translates 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. 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: CanTpTxSduId and CanTpTxInfoPtr. Note: only the length information within the CanTpTxInfoPtr structure shall be analyzed. The pointer to the payload data shall be discarded.
2	`PduR_CanTpCopyTxData ( SduId, PduInfoPtr, TxState, TxCntPtr )`	The PduR_CanTpCopyTxData is called to request the necessary transmit buffer. The upper layer copies segment data into the destination buffer.
3	`BUFREQ_OK`	The upper layer allocates and locks the required Tx buffer. Then returns `BUFREQ_OK`.
4	`CanIf_Transmit ( CanTxPduId, PduInfoPtr )`	Within the task, CanTp calls the CAN Interface by using CanIf_Transmit, where CanTxPduId 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( CanTpTxPduId, )`	CanTp awaits a confirmation from the CAN Interface (CanTp_TxConfirmation)
6	`PduR_CanTpCopyTxData ( SduId, PduInfoPtr, TxState, TxCntPtr )`	For each consecutive frame CanTp asks PDU Router for the buffer with new data to be sent.
7	`PduR_CanTpTxConfirmation ( CanTpTxSduId, Result )`	When all data have been sent, or when an error occurs, CanTp notifies PDU Router with PduR_CanTpTxConfirmation. CanTpTxPduId 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 is larger than 7 bytes (6 bytes in case of extended addressing format);
- Reception is successfully processed.

9.5.2 Sequence diagram



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 ( CanTpRxPduId, CanTpRxPduPtr )`	When the CAN Interface receives a frame (here a first frame), it notifies CanTp by means of a CanTp_RxIndication callback. CanTpRxPduId represents the ID of L-PDU that has been received and CanTpRxPduPtr indicates payload and L-SDU datalength to the L-SDU.
2	`PduR_CanTpStartOfReception( CanTpRxSduId, TpSduLength, PduInfoPtr )`	CanTp ask PDU Router to provide a buffer 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 ContinueToSend. (see step 8.)
5	`CanTp_RxIndication ( CanTpRxPduId, CanTpRxPduPtr )`	When the CAN Interface receives a frame (here a consecutive frame), CAN Interface notifies CanTp by means of a CanTp_RxIndication callback. CanTpRxPduId represents the ID of the CAN frame that has been received and CanTpRxPduPtr indicates payload to the L-SDU.
6		CanTp shall verify the sequence number and if correct, it asks the PduR to copy the data to the buffer provided.

7	`PduR_CanTpCopyRxData( PduIdType, PduInfoType*, PduLengthType )`
	
	`Or`
	
	`PduR_CanTpRxIndication ( CanTpRxSduId, Result )`
		Three cases can append :
		
		- Normal Case: the buffer is not full, and the received consecutive frame is not the last one. CanTp has nothing special to do.
		
		- Buffer Full: the buffer provided is full. CanTp shall send a wait flow control and ask again a new buffer to PDU-Router. If there are extra bytes from the last CF, they have to be stored in this new buffer.
		
		- 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.

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.

10 Configuration specification

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

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

Chapter 10.3 specifies published information for the module CAN Transport Layer

[CANTP146] ⌈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.⌋(BSW159)

[CANTP147] ⌈The consistency of the configuration must be checked by the configuration tool at configuration time.⌋(BSW167)

[CANTP301] ⌈Configuration rules and constraints for plausibility checks will be performed where possible, during configuration time.⌋( )

10.1 How to read this chapter

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

- AUTOSAR Layered Software Architecture [2]
- AUTOSAR ECU Configuration Specification [4]. This document describes the AUTOSAR configuration methodology and the AUTOSAR configuration metamodel in detail.

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

10.1.1 Configuration and configuration parameters

Configuration parameters define the variability of the generic part(s) of an implementation of a module. This means that only generic or configurable module implementation can be adapted to the environment (software/hardware) in use during system and/or ECU configuration.
The configuration of parameters can be achieved at different times during the software process: before compile time, before link time or after build time. In the following, the term “configuration class” (of a parameter) will be used in order to refer to a specific point in time during configuration.

10.1.2 Variants

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

10.1.3 Containers

Containers structure the set of configuration parameters. This means:

- *all* configuration parameters are kept in containers.
- (sub-) containers can reference (sub-) containers. It is possible to assign a multiplicity to these references. This multiplicity defines the possible number of occurrences of the contained parameters.

10.1.4 Specification template for configuration parameters

The following tables consist of three sections:

- general section
- configuration parameter section
- section of included/referenced containers

SWS Item	
Container Name	Identifies the container with a name
Description	Explains the intention and content of the container.
Configuration Parameters	

Name	Identifies the parameter by name.		
Description	Explains the intention of the configuration parameter.		
Type or Unit	Specifies the type of parameter (e.g., `uint8..uint32`) or specifies the unit of the parameter (e.g., `ms`)		
Range	Specifies the range (or possible values) of the parameter (e.g., `1..15`, `ON,OFF`)	Describes the value(s) or range(s).	
Configuration Class	***Pre-compile***	see[3]	Refer here to (a) variant(s).
	Link time	see[4]	Refer here to (a) variant(s).
	Post Build	see[5]	Refer here to (a) variant(s).
Scope	Describes the scope of the parameter. The scope describes the impact of the configuration parameter: Does the setting affect only one instance of the module (instance), all instances of this module (module), the ECU or a network? Possible values of scope : instance, module, ECU, network		
Dependency	Describes the dependencies with respect to the scope.		

[3] see the explanation below this table - Pre-compile time
[4] see the explanation below this table - Link time
[5] see the explanation below this table - Post Build

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
Reference a valid (sub)container by its name.	Specifies the number of possible instances of the referenced container and its contained configuration parameters. Possible values: <multiplicity> <min_multiplicity..max_multiplicity>	Describes the scope of the referenced sub-container. The scope describes the impact of the configuration parameter: Does the setting affect only one instance of the module (instance), all instances of this module (module), the ECU or a network? Possible values of scope : instance, module, ECU, network> Describes the dependencies with respect to the scope.

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

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

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

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

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

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

10.2 Containers and configuration parameters

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

10.2.1 Variants

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

10.2.2 CanTp

Module Name	*CanTp*
Module Description	Configuration of the CanTp (CAN Transport Protocol) module.

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CanTpConfig	1	This container contains the configuration parameters and sub containers of the AUTOSAR CanTp module. This container is a MultipleConfigurationContainer, i.e. this container and its sub-containers exist once per configuration set.
CanTpGeneral	1	This container contains the general configuration parameters of the CanTp module.

10.2.3 CanTpConfig

SWS Item	**CanTp290_Conf :**
Container Name	CanTpConfig [Multi Config Container]
Description	This container contains the configuration parameters and sub containers of the AUTOSAR CanTp module. This container is a MultipleConfigurationContainer, i.e. this container and its sub-containers exist once per configuration set.
Configuration Parameters	

SWS Item	**CanTp240_Conf :**		
Name	CanTpMainFunctionPeriod {CANTP_MAIN_FUNCTION_PERIOD}		
Description	Allow to configure the time for the MainFunction (as float in seconds). 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 .. 0.255`		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***

CanTpChannel	1..*	This container contains the configuration parameters of the CanTp channel.

10.2.4 CanTpGeneral

SWS Item	**CanTp278_Conf :**
Container Name	CanTpGeneral{CanTpConfiguration}
Description	This container contains the general configuration parameters of the CanTp module.
Configuration Parameters	

SWS Item	**CanTp299_Conf :**		
Name	CanTpChangeParameterApi {CANTP_CHANGE_PARAMETER_API}		
Description	This parameter, if set to true, enables the CanTp_ChangeParameterRequest Api for this Module.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**CanTp239_Conf :**		
Name	CanTpDevErrorDetect {CANTP_DEV_ERROR_DETECT}		
Description	Switches the Development Error Detection and Notification ON or OFF		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**CanTp298_Conf :**		
Name	CanTpPaddingByte {CANTP_PADDING_BYTE}		
Description	Used for the initialization of unused bytes with a certain value		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**CanTp300_Conf :**		
Name	CanTpReadParameterApi {CANTP_READ_PARAMETER_API}		
Description	This parameter, if set to true, enables the CanTp_ReadParameterApi for this module.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants

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

SWS Item	**CanTp283_Conf :**		
Name	CanTpVersionInfoApi		
Description	The function CanTp_GetVersionInfo is configurable (On/Off) by this configuration parameter.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.2.5 CanTpChannel

SWS Item	**CanTp288_Conf :**
Container Name	CanTpChannel
Description	This container contains the configuration parameters of the CanTp channel.
Configuration Parameters	

SWS Item	**CanTp289_Conf :**		
Name	CanTpChannelMode		
Description	The CAN Transport Layer supports half and full duplex channel modes.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	CANTP_MODE_FULL_DUPLEX		Full duplex channel.
	CANTP_MODE_HALF_DUPLEX		Half duplex channel.
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

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.
CanTpTxNSdu	0..*	The following parameters needs to be configured for each CAN N-SDU that the CanTp module transmits via the CanTpChannel.

10.2.6 CanTpRxNSdu

SWS Item	**CanTp137_Conf :**
Container Name	CanTpRxNSdu{RxNsdu}
Description	The following parameters needs to be configured for each CAN N-SDU that the CanTp module receives via the CanTpChannel.

Configuration Parameters

SWS Item	CanTp276_Conf :		
Name	CanTpBs {CANTP_BS}		
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	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	CanTp277_Conf :		
Name	CanTpNar {CANTP_NAR}		
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	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	CanTp245_Conf :		
Name	CanTpNbr {CANTP_NBR}		
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	1		
Type	EcucFloatParamDef		
Range	`0 .. INF`		
Default value	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	CanTp279_Conf :	
Name	CanTpNcr {CANTP_NCR}	
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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**CanTp281_Conf :**		
Name	CanTpRxAddressingFormat {CANTP_RX_ADDRESSING_FORMAT}		
Description	Declares which communication addressing mode is supported for this Rx N-SDU. Enum values: CanTpStandard. To use normal addressing format. CanTpExtended. To use extended addressing format. CanTpMixed. To use mixed addressing format.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	CANTP_EXTENDED		Extended addressing format
	CANTP_MIXED		Mixed addressing format
	CANTP_STANDARD		Standard addressing format
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**CanTp280_Conf :**		
Name	CanTpRxDl {CANTP_DL}		
Description	Data Length Code of this RxNsdu. In case of variable message length, this value indicates the minimum data length. Depending on SF or FF N-SDU the value will be limited to 7 (6 for an extended addressing format) and 4095 respectively.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**CanTp301_Conf :**		
Name	CanTpRxNSduId {CANTP_RXNSDU_ID}		
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	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	CanTp249_Conf :		
Name	CanTpRxPaddingActivation {CANTP_PADDING_ACTIVATION}		
Description	Defines if the receive frame uses padding or not. 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) CanTpOff: The N-PDU received does not use padding for SF, CF and the last CF. (N-PDU length is dynamic)		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	CANTP_OFF	Padding is not used	
	CANTP_ON	Padding is used	
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	CanTp250_Conf :		
Name	CanTpRxTaType {CANTP_TA_TYPE}		
Description	Declares the communication type of this Rx N-SDU.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	CANTP_FUNCTIONAL	Functional request type	
	CANTP_PHYSICAL	Physical request type	
ConfigurationClass	Pre-compile time	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	CanTp251_Conf :		
Name	CanTpRxWftMax {CANTP_WFTMAX}		
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	--		
ConfigurationClass	Pre-compile time	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	CanTp252_Conf :
Name	CanTpSTmin {CANTP_STMIN}
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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**CanTp241_Conf :**		
Name	CanTpRxNSduRef		
Description	Reference to a Pdu in the COM-Stack.		
Multiplicity	1		
Type	Reference to [Pdu]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CanTpNAe	0..1	Contains the parameters needed to configure each RxNSdu or TxNSdu with CanTpAddressingFormat set to CanTpMixed.
CanTpNSa	0..1	Contains the parameters needed to configure each RxNSdu or TxNSdu with CanTpAddressingFormat set to CanTpExtended.
CanTpNTa	0..1	The following parameters need to be configured for each RxNsdu or TxNsdu with the CanTpAddressingFormat set to CanTpExtended.
CanTpRxNPdu	1	Used for grouping of the ID of a PDU and the Reference to a PDU.
CanTpTxFcNPdu	0..1	Used for grouping of the ID of a PDU and the Reference to a PDU.

10.2.7 CanTpRxNPdu

SWS Item	**CanTp256_Conf :**
Container Name	CanTpRxNPdu
Description	Used for grouping of the ID of a PDU and the Reference to a PDU.
Configuration Parameters	

SWS Item	**CanTp258_Conf :**		
Name	CanTpRxNPduId {CANTP_RXNPDU_ID}		
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	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CanTp257_Conf :**		
Name	CanTpRxNPduRef		
Description	Reference to a Pdu in the COM-Stack.		
Multiplicity	1		
Type	Reference to [Pdu]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

No Included Containers

10.2.8 CanTpTxFcNPdu

SWS Item	**CanTp259_Conf :**
Container Name	CanTpTxFcNPdu
Description	Used for grouping of the ID of a PDU and the Reference to a PDU.
Configuration Parameters	

SWS Item	**CanTp287_Conf :**		
Name	CanTpTxFcNPduConfirmationPduId		
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	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**CanTp260_Conf :**		
Name	CanTpTxFcNPduRef		
Description	Reference to a Pdu in the COM-Stack.		
Multiplicity	1		
Type	Reference to [Pdu]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

No Included Containers

10.2.9 CanTpTxNSdu

SWS Item	**CanTp138_Conf :**
Container Name	CanTpTxNSdu{TxNsdu}
Description	The following parameters needs to be configured for each CAN N-SDU that the CanTp module transmits via the CanTpChannel.
Configuration Parameters	

SWS Item	**CanTp263_Conf :**
Name	CanTpNas {CANTP_NAS}

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

SWS Item	**CanTp264_Conf :**		
Name	CanTpNbs {CANTP_NBS}		
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	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**CanTp265_Conf :**		
Name	CanTpNcs {CANTP_NCS}		
Description	Value in seconds of the performance requirement of (N_Cs + N_As). N_Cs is the time which elapses between the transmit request of a CF N-PDU until the transmit request of the next CF N-PDU.		
Multiplicity	0..1		
Type	EcucFloatParamDef		
Range	`0 .. INF`		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**CanTp282_Conf :**		
Name	CanTpTc {CANTP_TC}		
Description	switch for enabling Transmit Cancellation.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**CanTp262_Conf :**
Name	CanTpTxAddressingFormat {CANTP_TX_ADDRESSING_FORMAT}
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 (the N_TA container of this TxNsdu will be used). CanTpMixed to use mixed addressing format (the N_AE container of this TxNsdu will be used).		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	CANTP_EXTENDED	Extended addressing format	
	CANTP_MIXED	Mixed addressing format	
	CANTP_STANDARD	Standard addressing format	
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**CanTp267_Conf :**		
Name	CanTpTxDl {CANTP_DL}		
Description	Data Length Code of this TxNsdu. In case of variable length message, this value indicates the minimum data length.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**CanTp268_Conf :**		
Name	CanTpTxNSduId {CANTP_TXNSDU_ID}		
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	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**CanTp269_Conf :**	
Name	CanTpTxPaddingActivation {CANTP_PADDING_ACTIVATION}	
Description	Defines if the transmit frame use padding or not. 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) CanTpOff The transmit N-PDU does not use padding for SF, CF and the last CF. (N-PDU length is dynamic)	
Multiplicity	1	
Type	EcucEnumerationParamDef	
Range	CANTP_OFF	Padding is not used
	CANTP_ON	Padding is used

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

SWS Item	**CanTp270_Conf :**		
Name	CanTpTxTaType {CANTP_TA_TYPE}		
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
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**CanTp261_Conf :**		
Name	CanTpTxNSduRef		
Description	Reference to a Pdu in the COM-Stack.		
Multiplicity	1		
Type	Reference to [Pdu]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
CanTpNAe	0..1	Contains the parameters needed to configure each RxNSdu or TxNSdu with CanTpAddressingFormat set to CanTpMixed.
CanTpNSa	0..1	Contains the parameters needed to configure each RxNSdu or TxNSdu with CanTpAddressingFormat set to CanTpExtended.
CanTpNTa	0..1	The following parameters need to be configured for each RxNsdu or TxNsdu with the CanTpAddressingFormat set to CanTpExtended.
CanTpRxFcNPdu	0..1	Used for grouping of the ID of a PDU and the Reference to a PDU.
CanTpTxNPdu	1	Used for grouping of the ID of a PDU and the Reference to a PDU.

10.2.10 CanTpTxNPdu

SWS Item	**CanTp274_Conf :**
Container Name	CanTpTxNPdu
Description	Used for grouping of the ID of a PDU and the Reference to a PDU.
Configuration Parameters	

SWS Item	**CanTp286_Conf :**
Name	CanTpTxNPduConfirmationPduId
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	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**CanTp275_Conf :**		
Name	CanTpTxNPduRef		
Description	Reference to a Pdu in the COM-Stack.		
Multiplicity	1		
Type	Reference to [Pdu]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

No Included Containers

10.2.11 CanTpRxFcNPdu

SWS Item	**CanTp271_Conf :**
Container Name	CanTpRxFcNPdu
Description	Used for grouping of the ID of a PDU and the Reference to a PDU.
Configuration Parameters	

SWS Item	**CanTp273_Conf :**		
Name	CanTpRxFcNPduId {CANTP_RXFC_NPDU_ID}		
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	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**CanTp272_Conf :**		
Name	CanTpRxFcNPduRef		
Description	Reference to a Pdu in the COM-Stack.		
Multiplicity	1		
Type	Reference to [Pdu]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

No Included Containers

10.2.12 CanTpNTa

SWS Item	**CanTp139_Conf :**
Container Name	CanTpNTa{N_Ta}
Description	The following parameters need to be configured for each RxNsdu or TxNsdu with the CanTpAddressingFormat set to CanTpExtended.
Configuration Parameters	

SWS Item	**CanTp255_Conf :**		
Name	CanTpNTa {CANTP_NTA}		
Description	If an RxNsdu or a TxNsdu is configured for extended addressing format, this parameter contains the transport protocol target address's value.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	--		
ConfigurationClass	*Pre-compile time*	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

No Included Containers

10.2.13 CanTpNSa

SWS Item	**CanTp253_Conf :**
Container Name	CanTpNSa{N_Sa}
Description	Contains the parameters needed to configure each RxNSdu or TxNSdu with CanTpAddressingFormat set to CanTpExtended.
Configuration Parameters	

SWS Item	**CanTp254_Conf :**		
Name	CanTpNSa {CANTP_NSA}		
Description	If an RxNSdu or a TxNSdu is configured for extended addressing format, this parameter contains the transport protocol source address's value.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	--		
ConfigurationClass	*Pre-compile time*	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.2.14 CanTpNAe

SWS Item	**CanTp284_Conf :**
Container Name	CanTpNAe{N_Ae}
Description	Contains the parameters needed to configure each RxNSdu or TxNSdu with CanTpAddressingFormat set to CanTpMixed.

Configuration Parameters

SWS Item	**CanTp285_Conf :**		
Name	CanTpNAe {CANTP_NAE}		
Description	If an RxNsdu or a TxNsdu is configured for mixed addressing format, this parameter contains the transport protocol address extension value.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

No Included Containers

10.3 Published Information

[CANTP326] ⌈The standardized common published parameters as required by BSW00402 in the General Requirements on Basic Software Modules [3] shall be published within the header file of this module and need to be provided in the BSW Module Description. The according module abbreviation can be found in the List of Basic Software Modules [1].⌋( )

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

11 Changes to Release 4 Rev. 2

11.1 Deleted SWS Items

SWS Item	*Rationale*
CANTP227	
CanTp291_Conf	CANTP_E_COM and CANTP_E_OPER_NOT_SUPPORTED production error, to development error.
CanTp292_Conf	CANTP_E_COM production error to development error.
CanTp297_Conf	CANTP_E_OPER_NOT_SUPPORTED production error to development error.
CANTP130	
CANTP192	
CANTP292	
CANTP100	
CanTp293_Conf	
CanTp295_Conf	
CANTP222	
CANTP276	

11.2 Changed SWS Items

SWS Item	*Rationale*
CANTP320	Invalid API name in specification requirement.
CANTP321	Invalid API name in specification requirement.
CANTP322	Invalid API name in specification requirement.
CANTP272	TP_NORETRY enum value not needed anymore
CanTp277_Conf	Change range of EcucFloatParamDef from –INF.. INF to 0..INF
CanTp245_Conf	Change range of EcucFloatParamDef from –INF.. INF to 0..INF
CanTp279_Conf	Change range of EcucFloatParamDef from –INF.. INF to 0..INF
CanTp263_Conf	Change range of EcucFloatParamDef from –INF.. INF to 0..INF
CanTp264_Conf	Change range of EcucFloatParamDef from –INF.. INF to 0..INF
CanTp265_Conf	Change range of EcucFloatParamDef from –INF.. INF to 0..INF
CANTP166	Remove wrong sentence from the requirement.
CANTP082	Requirement clarification.
CANTP254	Change relevance of CANTP_E_OPER_NOT_SUPPORTED error from production to development.
CANTP260	Change relevance of CANTP_E_OPER_NOT_SUPPORTED error from production to development.
CANTP283	Requirement clarification.
CANTP264	Dem.h is not included anymore, because CanTp does not report production errors.
CANTP229	
CANTP209	
CANTP216	
CanTp276_Conf	
CANTP314	

11.3 Added SWS Items

SWS Item	*Rationale*
CanTp301_Conf	Parameter needed by CanTp_CancelReceive, CanTp_ChangeParameter, CanTp_ReadParameter
CANTP325	Impact of constant block size parameter.

12 Not applicable requirements

[CANTP327] ⌈These requirements are not applicable to this specification.⌋ (BSW00344, BSW00404, BSW00405, BSW170, BSW00419, BSW382, BSW383, BSW397, BSW398, BSW399, BSW400, BSW00375, BSW00416, BSW168, BSW00423, BSW00427, BSW00428, BSW00429, BSW00431, BSW00432, BSW00433, BSW00434, BSW00422, BSW00420, BSW00417, BSW161, BSW162, BSW00324, BSW00415, BSW00325, BSW00326, BSW00342, BSW00413, BSW00347, BSW00307, BSW00314, BSW00361, BSW00328, BSW00378, BSW172, BSW010, BSW00321, BSW00341, BSW00334)