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

Document Version	2.6.0
Document Status	Final
Part of Release	3.2
Revision	3

Document Change History

Date	Version	Changed by	Change Description
28.02.2014	2.6.0	AUTOSAR Release Management	• Added behavior when CanIf_Transmit returns E_NOT_OK • Clarification on the PADDING mechanism • Clarification on the sending of FC(OVFLW) and FC(WT) • Clarification for reaction on BUFREQ_E_BUSY • Editorial changes • Removed chapter(s) on change documentation
17.05.2012	2.5.0	AUTOSAR Administration	• Update the CancelTransmit/CancelReceive mechanism
27.04.2011	2.4.0	AUTOSAR Administration	• CanTp supports Full Duplex Mode • Added possibility to change and read CanTp parameters • Changed production errors to development errors • BS and STMIN are not changed during a reception session. • Legal disclaimer revised
15.09.2010	2.3.0	AUTOSAR Administration	• Removed CanTp228 • Updated CanTp246, CanTp248 • Legal disclaimer revised
23.06.2008	2.2.1	AUTOSAR Administration	Legal disclaimer revised

Document Change History			
Date	**Version**	**Changed by**	**Change Description**
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
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.

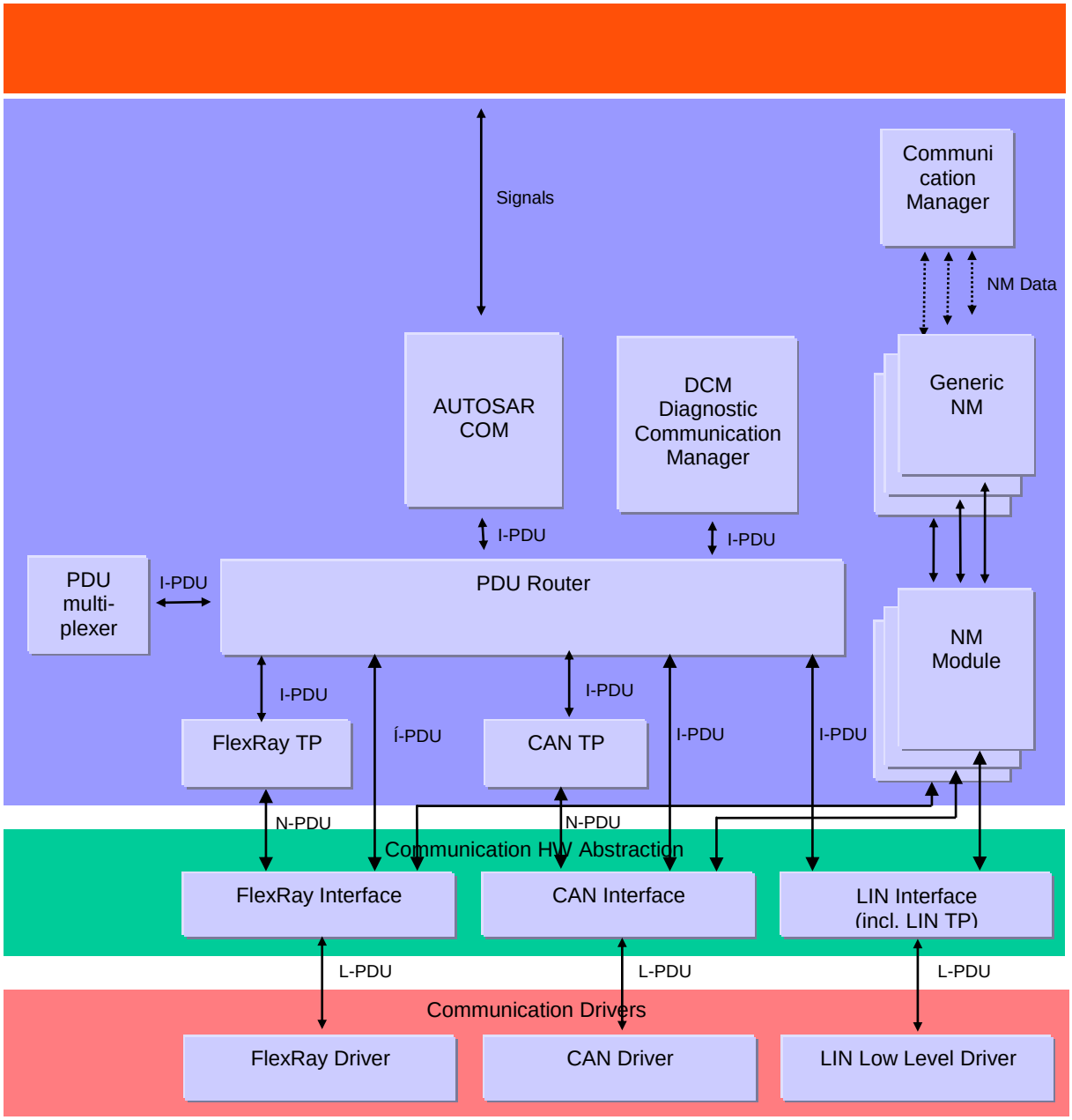


Figure 1 : AUTOSAR Communication Stack

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.

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 an 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).
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, N_TAtype and Mtype. N_TA is filed 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.
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.

Definitions:	*Description:*
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.
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.
Connection channel	The CAN Transport Layer is handling resources used by multiple connections in order to save RAM. When a connection becomes active, the channel that is used by this connection will be unavailable for other connections.
Connection	A transport protocol session, either is a transmission or a reception session on a N-SDU.

3 Related documentation

3.1 Input documents

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

[2] Layered Software Architecture,
AUTOSAR_LayeredSoftwareArchitecture.pdf

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

[4] Specification of ECU Configuration,
AUTOSAR_ECU_Configuration.pdf

[5] Glossary
AUTOSAR_Glossary.pdf

[6] Requirements on CAN
AUTOSAR_SRS_CAN.pdf

[7] Specification of CAN Interface
AUTOSAR_SWS_CAN_Interface.pdf

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

[9] Specification of Function Inhibition Manager
AUTOSAR_SWS_FIM.pdf

[10] Specification of PDU Router
AUTOSAR_SWS_PDU_Router.pdf

[11] Specification of Diagnostic Event Manager (DEM)
AUTOSAR_SWS_DEM.pdf

[12] AUTOSAR Basic Software Module Description Template,
AUTOSAR_BSW_Module_Description.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[1]. 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.

[1] Usage of CAN Transport Layer for AUTOSAR COM I-PDUs is planned for AUTOSAR Phase 2.

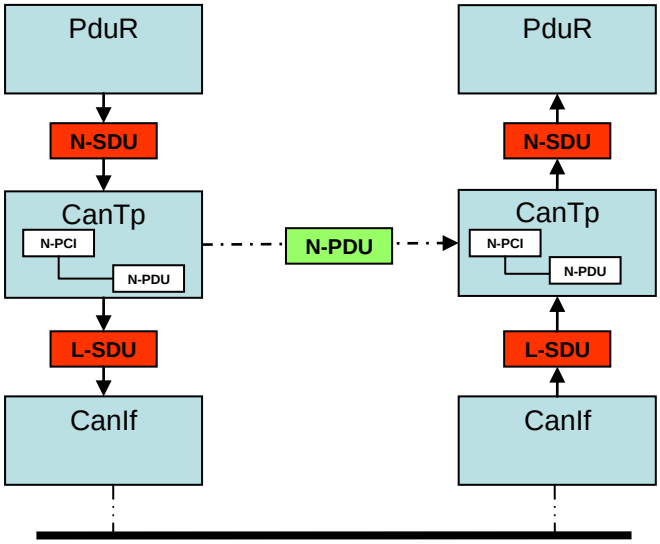


Figure 2: CAN Transport Layer interactions

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 (e.g. DCM 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)
- Addressing type of each CAN N-SDU (physical or functional)

- Addressing format of each connection (standard or extended) and, in the case of extended addressing format, the N_TA 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

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 and notify reception of a message from/to the PDU-Router, and request a buffer, to reassemble segmented frames:

- *PduR_CanTpRxIndication*
- *PduR_CanTpProvideRxBuffer*
- *PduR_CanTpProvideTxBuffer*
- *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. At this point it should be noted, that the code-file structure should include the following files:

- CanTp_Lcfg.c – for parameters configurable at link time
- CanTp_PBcfg.c – for parameters, which are configurable post-build.

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 construct its include file structure as shown in Figure 3.

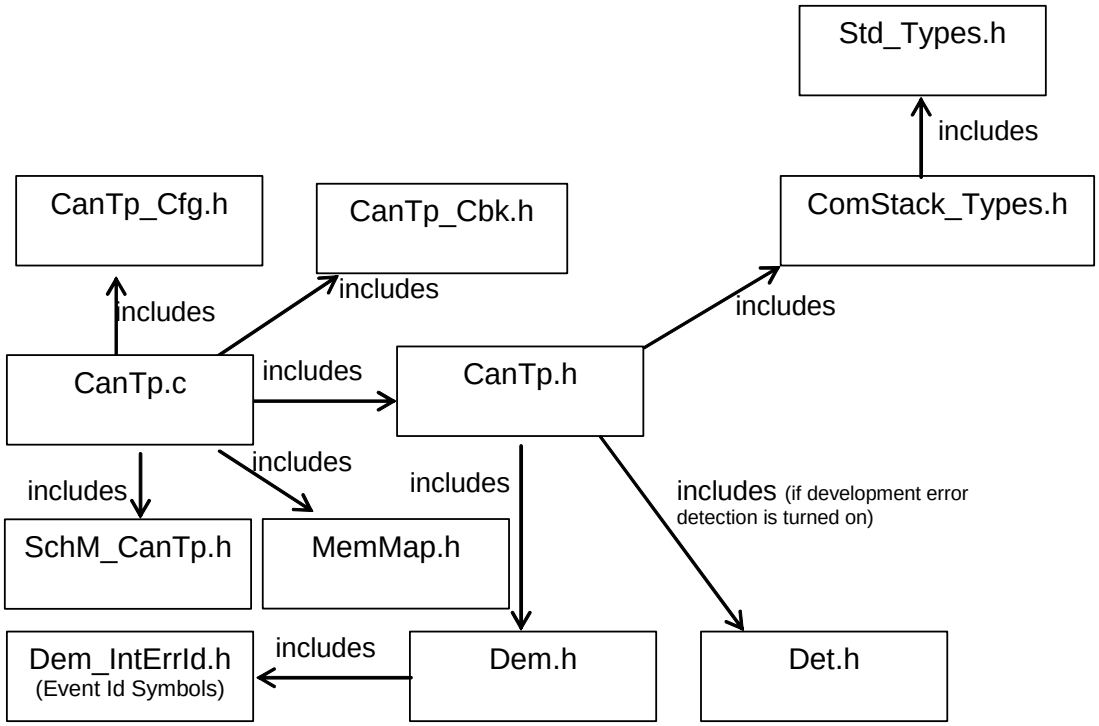


Figure 3: File Structure

Global data types and functions that are only used internally by the CAN Transport Protocol, are given in CanTp.c

CanTp219: CanTp.c shall include CanTp.h.

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.

CanTp001: `CanTp_Cfg.h` shall define constant and customizable data for module configuration at pre-compile time.

CanTp221: `CanTp.h` 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.

CanTp024: Each header and C file of the CanTp module shall provide the possibility of version identification of the CAN Transport module by CANTP_MAJOR_VERSION, CANTP_MINOR_VERSION and CANTP_PATCH_VERSION.

Version number macros can then be used for checking and reading out the software version of a software module, during compile-time and run-time.

The module could include the Dem.h file. By this inclusion the APIs to report errors as well as the required Event Id symbols are included.
Although this specification does not define any production errors, the eventually name of the Event Id symbols are provided by XML to the DEM configuration tool. The DEM configuration tool assigns ECU dependent values to the Event Id symbols and publishes the symbols in Dem_IntErrId.h.

5.2.3 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.

CanTp151: The CanTp module's source shall not use compiler and platform specific keywords

CanTp152: The CanTp module's source shall indicate all global data with read-only properties by explicitly assigning the keyword `const`.

CanTp153: The CanTp module may use macros (instead of functions) where source code is used and runtime is critical.

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)

CanTp158: The CanTp module's source shall not be processor and compiler dependent.

6 Requirements traceability

Document: General Requirements on Basic Software Modules [3]

Functional general requirements	
Requirement	***Satisfied by***
[BSW00344] Reference to link-time configuration	Not applicable (This module does not use Link Time configuration parameters)
[BSW00404] Reference to post build time configuration	Not applicable (requirement on implementation, not on specification)
[BSW00405] Reference to multiple configuration sets	Not applicable (This module does not use multiple configuration sets)
[BSW00345] Pre-Build Configuration	CanTp001 chapter 10
[BSW159] Tool-based configuration	CanTp146
[BSW167] Static configuration checking	CanTp147
[BSW171] Configurability of optional functionality	chapter 10
[BSW170] Data for reconfiguration of SW-components	Not applicable. (Requirement on SWC module)
[BSW380] Separate C-File For configuration parameters	CanTp159
[BSW00419] Separate C-Files for pre-compile time configuration parameters	Not applicable (No “const” pre-compile time parameter)
[BSW381] Separate configuration header file for pre-compile time parameters	CanTp001
[BSW412] Separate H-File for configuration parameters	CanTp156
[BSW382] Not-used configuration elements need to be listed	Not applicable (there are no not-used configuration elements for this module)
[BSW383] List dependencies of configuration files	Not applicable (this module does not use configuration files from other modules)
[BSW384] List dependencies to other modules	Fulfilled by chapter 5
[BSW385] List possible error notifications	CanTp101
[BSW386] Configuration for detecting an error	CanTp101
[BSW387] Specify the configuration class of callback function	Fulfilled by chapter 8.4
[BSW388] Introduce containers	Fulfilled by configuration chapter 10
[BSW389] Containers shall have names	Fulfilled by configuration chapter 10
[BSW390] Parameter content shall be unique within the module	Fulfilled by configuration chapter 10
[BSW391] Parameter shall have unique names	Fulfilled by configuration chapter 10
[BSW392] Parameters shall have a type	Fulfilled by configuration chapter 10
[BSW393] Parameters shall have a range	Fulfilled by configuration chapter 10
[BSW394] Specify the scope of the parameters	Fulfilled by configuration chapter 10
[BSW395] List the required parameters (per parameter)	Fulfilled by configuration chapter 10
[BSW396] Configuration classes	Fulfilled by configuration chapter 10
[BSW397] Pre-compile-time parameters	Not applicable (definition)
[BSW398] Link-time parameters	Not applicable (definition)

[BSW399] Loadable Post-build time parameters	Not applicable (definition)
[BSW400] Selectable Post-build time parameters	Not applicable (definition)
[BSW402] Published information	CanTp140
[BSW00375] Notification of wake-up reason	Not applicable (this module does not provide any reason for wake-up)
[BSW101] Initialization interface	CanTp208
[BSW00416] Sequence of Initialization	Not applicable (requirement on system design, not on a single module)
[BSW406] Check module initialization	CanTp161
[BSW168] Diagnostic Interface of SW components	Not applicable (this module does not support a special diagnostic interface)
[BSW407] Function to read out published parameters	CanTp162 CanTp163
[BSW00423] Usage of SW-C template to describe BSW modules with AUTOSAR Interfaces	Not applicable. (This module has no interface with RTE)
[BSW00424] BSW main processing function task allocation	CanTp164
[BSW00425] Trigger conditions for schedulable objects	Not covered. New template needed
[BSW00426] Exclusive areas in BSW modules	Not covered. New template needed
[BSW00427] ISR description for BSW modules	Not applicable. (this module does not provide any ISRs)
[BSW00428] Execution order dependencies of main processing functions	Not applicable. (This module has only 1 MainFunction)
[BSW00429] Restricted BSW OS functionality access	Not applicable (this module doesn't use any OS objects or services)
[BSW00431] The BSW Scheduler module implements task bodies	Not applicable (requirement on the BSW scheduler module)
[BSW00432] Modules should have separate main processing functions for read/receive and write/transmit data path	Not applicable. (Mainfunction is used to manage time)
[BSW00433] Calling of main processing functions	Not applicable (requirement on the BSW scheduler module)
[BSW00434] The Schedule Module shall provide an API for exclusive areas	Not applicable (requirement on the BSW scheduler module)
[BSW00336] Shutdown interface	CanTp010
[BSW00337] Classification of errors	CanTp101
[BSW00338] Detection and Reporting of development errors	
[BSW00369] Do not return development error codes via API	CanTp021
[BSW00339] Reporting of production relevant errors and exceptions	Not applicable (no production errors)
[BSW00421] Reporting of production relevant error events	Not applicable (no production errors)
[BSW00422] Debouncing of production relevant error status	Not applicable (DEM requirement)
[BSW00420] Production relevant error event rate detection	Not applicable (DEM requirement)
[BSW00417] Reporting of Error Events by Non-Basic Software	Not applicable (This module is a BSW module)
[BSW00323] API parameter checking	CanTp132
[BSW004] Version check	CanTp024 CanTp140

[BSW00435] Header File Structure for the Basic Software Scheduler	CanTp156
[BSW00436] Module Header File Structure for the Basic Software Memory Mapping	CanTp156

Non-functional general requirements	
Software Architecture Requirements	
Requirement	***Satisfied by***
[BSW161] Microcontroller abstraction	Not applicable (requirement on AUTOSAR architecture, not a single module)
[BSW162] ECU layout abstraction	Not applicable (requirement on AUTOSAR architecture, not a single module)
[BSW00324] Do not use HIS Library	Not applicable (requirement on AUTOSAR architecture, not a single module)
[BSW005] No hard coded horizontal interfaces within MCAL	See paragraph 5.1.6 & 5.1.7
[BSW00415] User dependent include files	Not applicable (no interface for specifics)
[BSW166] BSW Module interfaces	See paragraph 5.2
Software Integration Requirements	
Requirement	***Satisfied by***
[BSW164] Implementation of interrupt service routines	Fulfilled by API definitions in chapter 8
[BSW00325] Runtime of interrupt service routines	Not applicable (this module does not provide any ISRs)
[BSW00326] Transition from ISRs to OS tasks	Not applicable (this module does not provide any ISRs)
[BSW00342] Usage of source code and object code	Not applicable (requirement on implementation, not on specification)
[BSW00343] Specification and configuration of time	Fulfilled by configuration chapter 10
[BSW160] Human-readable configuration data	Fulfilled by configuration chapter 10
Software Module Design Requirements Software quality	
Requirement	***Satisfied by***
[BSW007] HIS MISRA C	CanTp150
Naming conventions	
Requirement	***Satisfied by***
[BSW00300] Module naming convention	Fulfilled by API definitions in chapter 8
[BSW00413] Accessing instances of BSW modules	Not applicable. (Only 1 instance of CanTp allowed)
[BSW00347] Naming separation of different instances of BSW drivers	Not applicable. (For driver only.)
[BSW00347] Naming separation of drivers	Not applicable (For driver only.)
[BSW00305] Self-defined data types naming convention	Fulfilled by type definitions in chapter 8
[BSW00307] Global variables naming convention	Not applicable (no global variables are specified for this module)
[BSW00310] API naming convention	CanTp003
[BSW00373] Main processing function naming convention	CanTp164
[BSW00327] Error values naming convention	CanTp101

[BSW00335] Status values naming convention	Fulfilled by API definitions in chapter 8
[BSW00350] Development error detection keyword	CanTp006
[BSW00408] Configuration parameter naming convention	Fulfilled by configuration chapter 10
[BSW00410] Compiler switches shall have defined values	Fulfilled by configuration chapter 10
[BSW00411] Get version info keyword	Fulfilled by configuration chapter 10

Module file structure

Requirement	***Satisfied by***
[BSW00346] Basic set of module files	CanTp156
[BSW158] Separation of configuration from implementation	CanTp156 CanTp001
[BSW00314] Separation of interrupt frames and service routines	Not applicable (this module does not provide any ISRs)
[BSW00370] Separation of callback interface from API	CanTp156

Standard header files

Requirement	***Satisfied by***
[BSW00348] Standard type header	See Section 8.1
[BSW00353] Platform specific type header	CanTp002
[BSW00361] Compiler specific language extension header	Not applicable (requirement on implementation, not on specification)

Module Design

Requirement	***Satisfied by***
[BSW00301] Limit imported information	CanTp156
[BSW00302] Limit exported information	CanTp157
[BSW00328] Avoid duplication of code	Not applicable (requirement on implementation, not on specification)
[BSW00312] Shared code shall be reentrant	Fulfilled by API definitions in chapter 8
[BSW006] Platform independency	CanTp158

Types and keywords

Requirement	***Satisfied by***
[BSW00357] Standard API return type	Fulfilled by API definitions in chapter 8
[BSW00377] Module Specific API return type	Fulfilled by API definitions in chapter 8
[BSW00304] AUTOSAR integer data types	Fulfilled by API definitions in chapter 8
[BSW00355] Do not redefine AUTOSAR integer data types	Fulfilled by API definitions in chapter 8
[BSW00378] AUTOSAR Boolean type	Not applicable (Not used)
[BSW00306] Avoid direct use of compiler and platform specific keywords	CanTp151

Global data

Requirement	Satisfied by
[BSW00308] Definition of global data	CanTp155
[BSW00309] Global data with read-only constraint	CanTp152

Interface and API

Requirement	Satisfied by
[BSW00371] Do not pass function pointers via API	Fulfilled by API definitions in chapter 8
[BSW00358] Return type of init() functions	CanTp208
[BSW00414] Parameter of init function	CanTp208
[BSW00376] Return type and parameters of main processing functions	CanTp164
[BSW00359] Return type of callback functions	Fulfilled by API definitions in chapter 8
[BSW00360] Parameters of callback functions	Fulfilled by API definitions in chapter 8
[BSW00329] Avoidance of generic interfaces	Fulfilled by API definitions in chapter 8
[BSW00330] Usage of macros instead of functions	CanTp153
[BSW00331] Separation of error and status values	Chapter 7.1.1 and CanTp101

Software Documentation Requirements

Requirement	Satisfied by
[BSW009] Module User Documentation	Fulfilled by the whole document
[BSW00401] Documentation of multiple instances of configuration parameters	Fulfilled by configuration chapter 10
[BSW172] Compatibility and documentation of scheduling strategy	Not applicable. (There is no scheduler in the CAN TP)
BSW010] Memory resource documentation	Not applicable. (requirement on implementation, not on specification)
[BSW00333] Documentation of callback function context	Fulfilled by API definitions in chapter 8
[BSW00374] Module vendor identification	CanTp140
[BSW00379] Module identification	CanTp140
[BSW003] Version identification	CanTp024 CanTp140
[BSW00318] Format of module version	
[BSW00321] Enumeration of module version numbers	Not applicable. (requirement on implementation, not on specification)
[BSW00341] Microcontroller compatibility documentation	Not applicable. (requirement on implementation, not on specification)
[BSW00334] Provision of XML file	Not applicable. (requirement on implementation, not on specification)

Document: AUTOSAR requirements on Basic Software, cluster CAN

Requirement	***Satisfied by***
[BSW01065] Usage of ISO 15765-2 specifications	CanTp033
[BSW01065] Usage of ISO 15765-4 specifications	See Section 7
[BSW01066] Concurrent connection configuration	CanTp096 CanTp120 CanTp121 CanTp122 CanTp123 CanTp124
[BSW01068] Unique identifier of N-SDU	CanTp035
[BSW01069] CAN address information and N-SDU identifier mapping	CanTp035
[BSW01071] Unique identifier of N-PDU	CanTp035
[BSW01073] Fixed N-PDU data length	CanTp116
[BSW01074] Transport connection properties	CanTp137 CanTp138
[BSW01075] CAN Transport Layer Initialization	CanTp170, CanTp030
[BSW01076] CAN Transport Layer Availability	CanTp031
[BSW01078] Support a subset of ISO 15765-2 addressing modes formats	CanTp035 CanTp137 CanTp138
[BSW01079] Compliance with CAN Interface notifications	CanTp019 CanTp020
[BSW01081] Connection specific timeout values	CanTp137 CanTp138
[BSW01082] Error handling	CanTp057
[BSW01086] Data value of unused bytes	CanTp059
[BSW01111] CAN Transport Layer Interfaces	This requirement is a specification recommendation fulfilled by chapter 8
[BSW01112] Independent interface	This requirement is a specification recommendation fulfilled by chapter 8
[BSW01116] Usage of different addressing modes formats in parallel	CanTp137 CanTp138 CanTp139
[BSW01117] Only half-duplex communication is supported	CanTp057
[BSW01149] Support of full-duplex communication	Chapter 4.1, CanTp057
[BSW01150] Selection of half/ full-duplex communication by configuration	CanTp289
[BSW01120] Multiple CAN Transport Layer instances	Multiple connections supported and therefore only one instance required.

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.

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`.

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()`.

CanTp238: The CanTp module shall performed 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`).

The COM Manager module should call the function `CanTp_Init()` before using the CanTp functionalities.

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.

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) and the module shall loose all current connections.

CanTp010: The function `CanTp_Shutdown` shall stop the CanTp module properly.

The COM Manager module shall call the function `CanTp_Shutdown()`.

The following figure summarizes all of the above requirements:

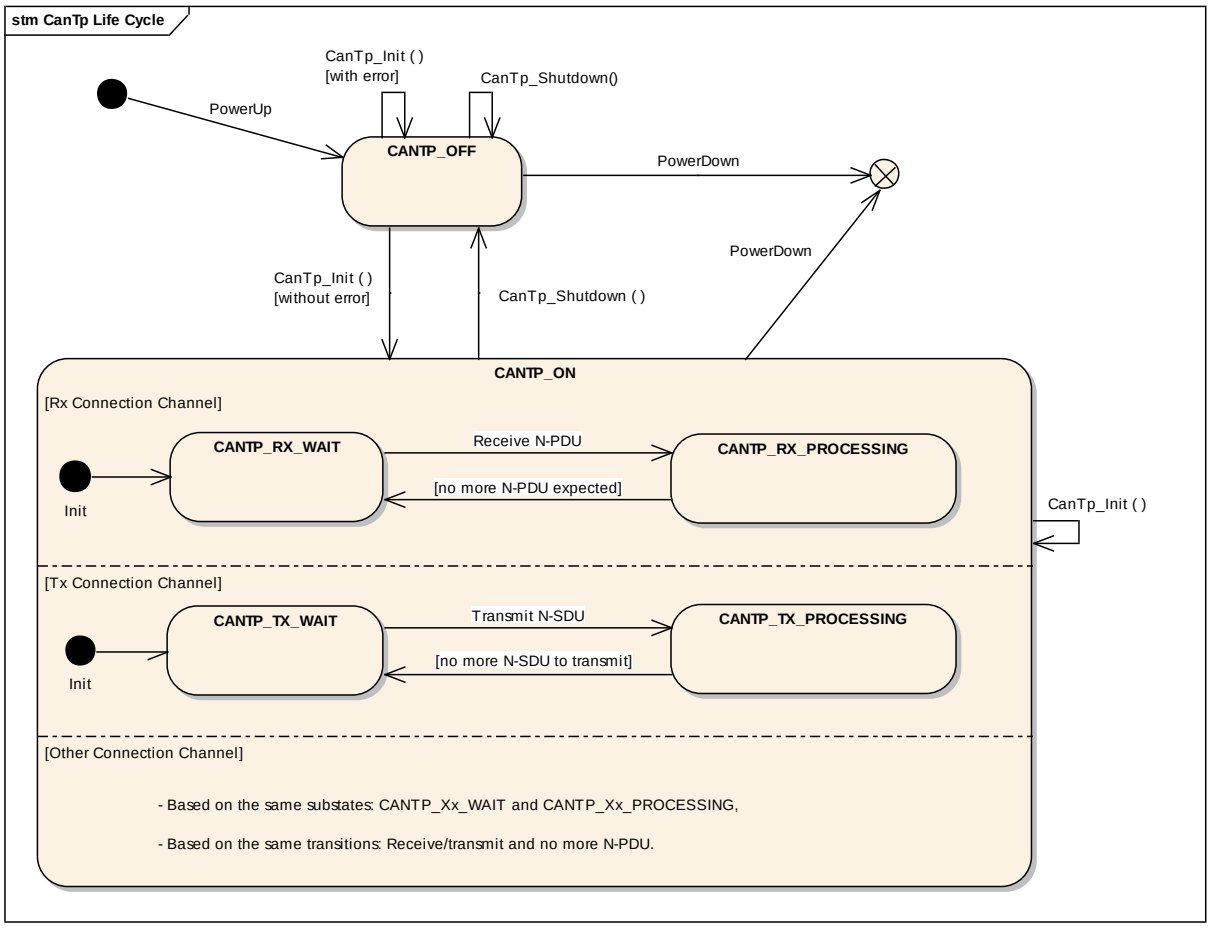


Figure 4: CAN Transport Layer life cycle

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).

CanTp072: The function CanTp_Transmit() shall reject the transmit request and return the status value E_NOT_OK if there is no hardware resource available.

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). 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. When the API function returns no transmission is in progress anymore.

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

CanTp074: 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

CanTp077: 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 diagnostic 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

To optimize communication stack resources, it has been decided to provide the CAN Transport Layer with limited buffering capacity.

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. These two operations shall be performed using the `PduR_CanTpProvideRxBuffer()` function.

CanTp080: The Rx buffer provided can be smaller than the expected N-SDU data length. In this case, when all blocks (defined by BS) that fit into the current buffer have been received, the CanTp module shall request another buffer by calling the `PduR_CanTpProvideRxBuffer()` service again.

To avoid confusion, it should be clarified that the expression “request a buffer” is not related to dynamic memory allocation. This expression simply means the upper layer makes a part of its internal buffer available to the CAN Transport Layer (i.e. the Rx buffer is locked until CanTp calls either `PduR_CanTpRxIndication()` or `PduR_CanTpProvideRxBuffer()`).

If the upper layer cannot provide a buffer 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_CanTpProvideRxBuffer()` function returns `BUFREQ_E_NOT_OK` or `BUFREQ_E_OVFL`.

CanTp081: After the reception of a First Frame, if the function `PduR_CanTpProvideRxBuffer` returns `BUFREQ_E_NOT_OK` to the CanTp module, the CanTp module shall abort the N-SDU reception. No Flow Control will be sent in this case.

CanTp326: After the reception of a First Frame, if the function `PduR_CanTpProvideRxBuffer` 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.

CanTp327: `PduR_CanTpRxIndication()` will not be called when the first `PduR_CanTpProvideRxBuffer()` returns either `BUFREQ_E_NOT_OK` or `BUFREQ_E_OVFL`.

CanTp315: If the function `PduR_CanTpProvideRxBuffer()` returns `BUFREQ_OK` to the CanTp module for the first call to this function after reception of a FF/SF, the CanTp shall copy the data of the SF/FF to the buffer, and call `PduR_CanTpProvideRxBuffer()` again.

CanTp330: If the function `PduR_CanTpProvideRxBuffer()` returns `BUFREQ_OK` to the CanTp module for the first call to this function after reception of a FF/SF but does not provide sufficient buffer for the CanTp to copy the data of the SF/FF, the CanTp module shall abort the N-SDU reception. No Flow Control will be sent in this case.

CanTp329: If the function `PduR_CanTpProvideRxBuffer()` returns another value than `BUFREQ_OK`, `BUFREQ_E_NOT_OK` or `BUFREQ_E_OVFL`, the CanTp module shall abort the reception. If the development error detection is enabled, a development error `CANTP_E_INVALID_BUFREQ` shall be triggered.

CanTp316: After reception of the last CF of the last block that fits completely into the provided buffer, CanTp shall call `PduR_CanTpProvideRxBuffer()` again.

CanTp317: In every call to `PduR_CanTpProvideRxBuffer()` following the first call after reception of a FF/SF, the CanTp shall use the parameter PduInfoPtr->SduLength to report the number of bytes actually copied to the last provided buffer.

CanTp082: If the buffer provided by the function `PduR_CanTpProvideRxBuffer()` to the CanTp module is insufficient for the next block, the CanTp module shall start a time-out N_Br, suspend the N-SDU reception, and call `PduR_CanTpProvideRxBuffer()` repeatedly from its MainFunction until N_Br expires or the buffer length is sufficient for the next block.

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 send a 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_NOT_OK occurs.

CanTp224: When the Rx buffer is finally provided, 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.

CanTp084: When the transport reception session is completed (successfully or not) the CanTp module shall call the upper layer notification service `PduR_CanTpRxIndication()`.

With regard to FF N-PDU reception, the content of the Flow Control N-PDU depends on the `PduR_CanTpProvideRxBuffer()` 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_CanTpProvideRxBuffer()` service.

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.

The BS value sent depends on the configuration parameter CanTpStaticBlockSize.

CanTp301: If the configuration parameter CanTpStaticBlockSize is set TRUE then the CanTp module shall use the same value for BS during the entire segmented reception. The BS will be calculated based on the result of the first call of `PduR_CanTpProvideRxBuffer()` service.

CanTp067: If the configuration parameter CanTpStaticBlockSize is set FALSE then CanTp could change the BS value during a segmented reception. If the Rx buffer provided is smaller than the entire N-SDU data length and its length is not a multiple of BS*7 (or BS*6 in case of extended addressing format), the CanTp module shall temporarily change the BS value to fill up the buffer reception completely. The desired behavior is described in the following picture:

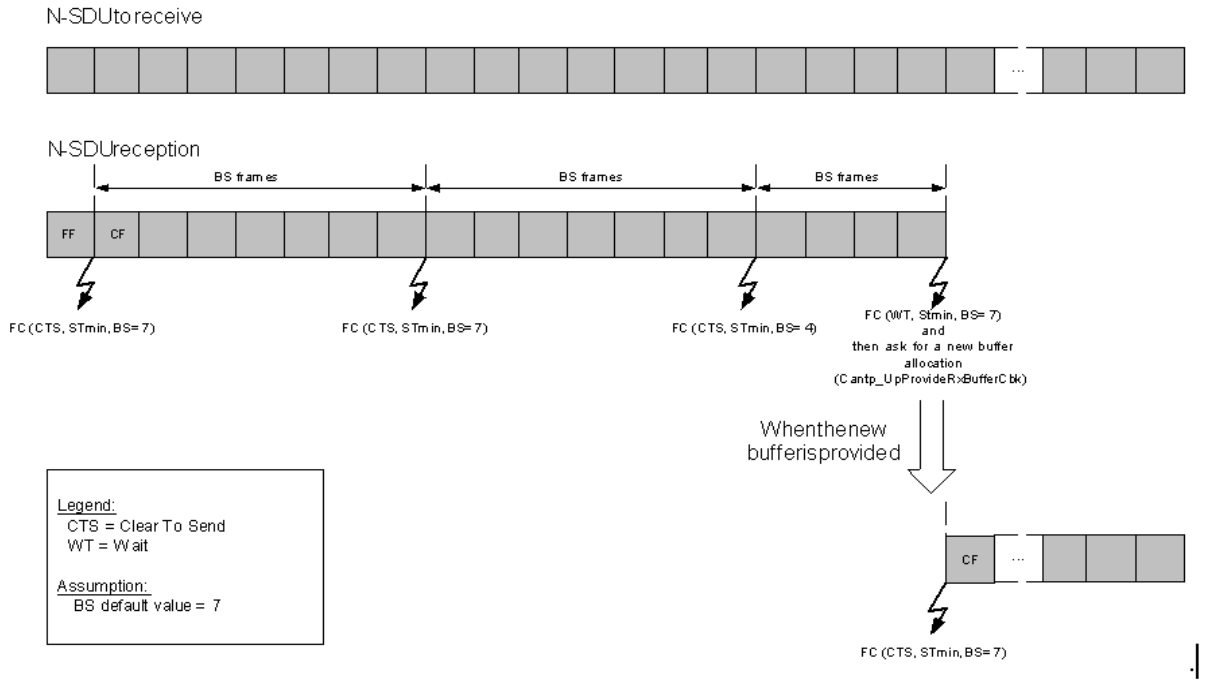


Figure 5: Management of the BS value

CanTp318: CanTp shall terminate the current reception connection when `CanIf_Transmit()` returns E_NOT_OK when transmitting an FC.

7.3.2 N-SDU Transmission

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

CanTp225: The function CanTp_Transmit shall only use the SduLength information within the PduInfoType parameter and shall not use the pointer to the payload N-SDU data.

CanTp226: After a transmission request from the upper layer, the CanTp module shall call `PduR_CanTpProvideTxBuffer()` at least once to request the necessary transmit data.

In the gateway case, the first call of `PduR_CanTpProvideTxBuffer()` will only provide the FF data, while the following calls of `PduR_CanTpProvideTxBuffer()` will always provide a complete block of data.

CanTp167: After a transmission request from upper layer, the CanTp module shall start time-out N_Cs before requesting Tx data. If data has not been provided before the timer elapsed, the CanTp module shall abort the communication.

The Tx data length provided can be smaller than the full Tx N-SDU data length.

CanTp086: If the data provided to the CanTp module is smaller than the full Tx N-SDU data and when the entire data provided has been sent, the CanTp module shall request data again by calling the function `PduR_CanTpProvideTxBuffer()`.

CanTp117: If the data provided cannot be sent completely, the CanTp layer shall request new data from the upper layer and shall store the remaining bytes that have not been sent yet.

If the upper layer cannot provide data because of an error (e.g. in the gateway case it may indicate that the transport session to the source network has been broken), the `PduR_CanTpProvideTxBuffer()` function returns `BUFREQ_E_NOT_OK`.

CanTp087: If `PduR_CanTpProvideTxBuffer()` 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.

If upper layer temporarily has no data available, the `PduR_CanTpProvideTxBuffer()` function returns `BUFREQ_E_BUSY`.

CanTp184: If the `PduR_CanTpProvideTxBuffer()` function returns `BUFREQ_E_BUSY`, the CanTp module shall later (implementation specific) retry to receive data for transmission.

CanTp185: If no data is provided 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 and notify the upper layer of this failure by calling the callback function `PduR_CanTpTxConfirmation` with the result `NTFRSLT_E_NOT_OK`.

The API `PduR_CanTpProvideTxBuffer()` contains a parameter length used for the recovery mechanism. Because ISO 15765-2 does not support such a mechanism, the CAN Transport Layer does not implement any kind of recovery. Thus, the length parameter is always set to zero (0) and upper layers can return data with any length.

CanTp186: The CanTp module shall set the length parameter in the call to `PduR_CanTpProvideTxBuffer` to zero (0).

CanTp089: When the Tx data is provided, the CanTp module shall resume the transmission of the N-SDU.

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.

CanTp319: CanTp shall terminate the current transmission connection when `CanIf_Transmit()` returns E_NOT_OK when transmitting an SF, FF, or CF.

7.3.3 Buffer strategy

Because CanTp has limited 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_CanTpProvideTxBuffer()` or `PduR_CanTpProvideRxBuffer()` 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.

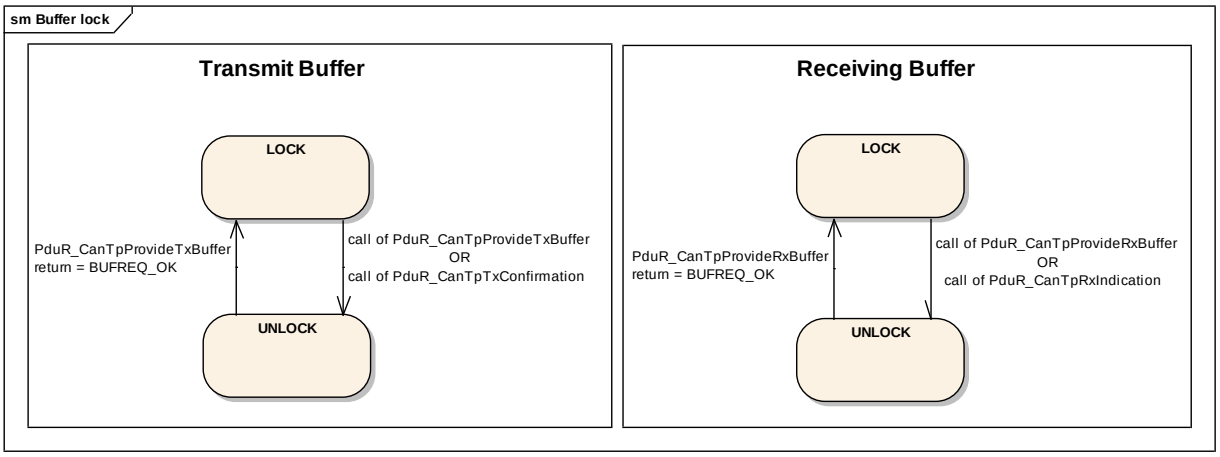
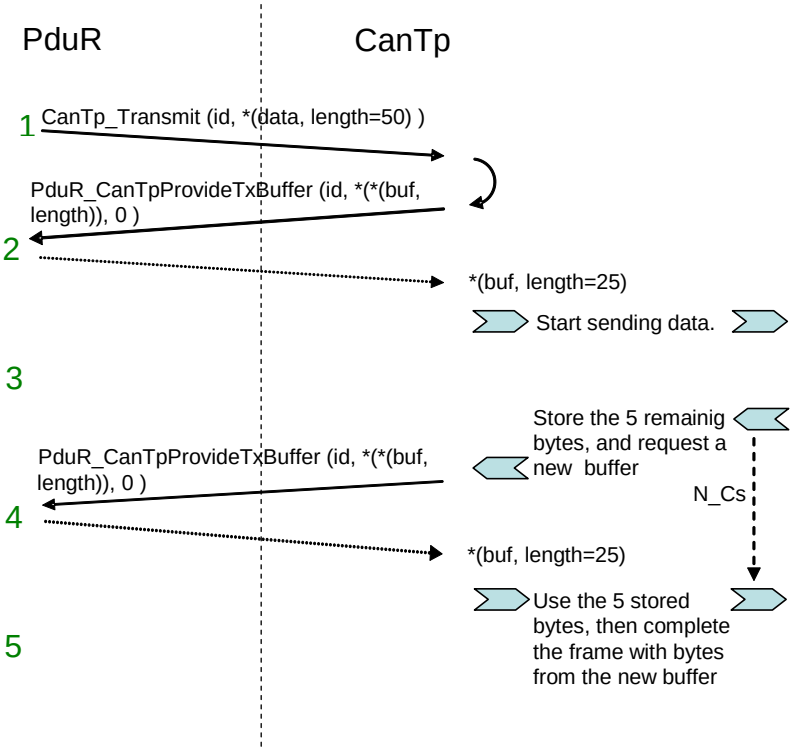


Figure 6: Tx and Rx Buffer locking

The PduR module will lock the buffer when it returns a status `BUFREQ_OK` to a `PduR_CanTpProvideTxBuffer()` or `PduR_CanTpProvideRxBuffer()` call and will keep the buffer locked until the CAN transport Layer requests a new buffer (`PduR_CanTpProvideTxBuffer()` or `PduR_CanTpProvideRxBuffer()` call) or when 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 and two sub-buffers of 25 bytes.



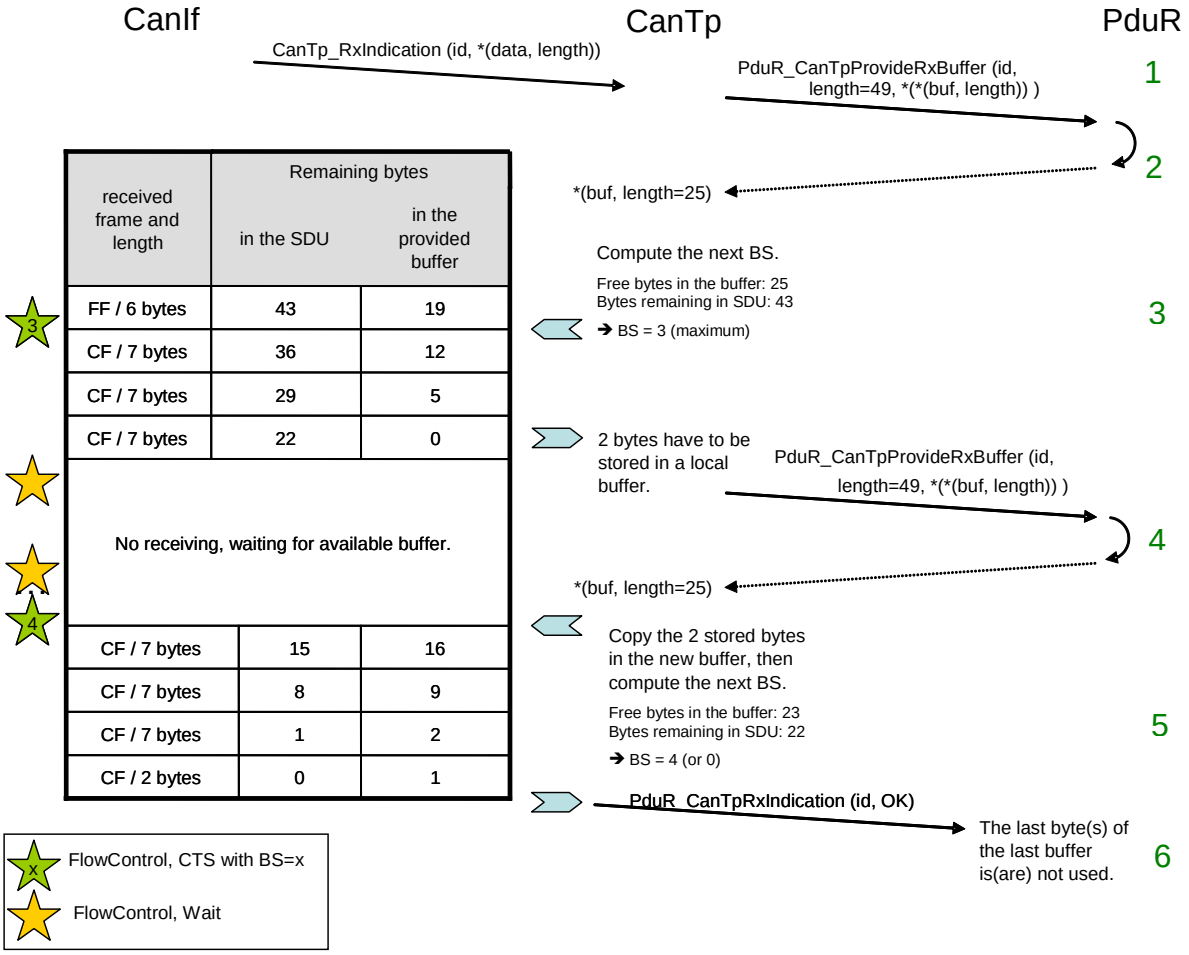
current frame and length	Remaining bytes in the SDU	Remaining bytes in the provided buffer
FF / 6 bytes	44	19
CF / 7 bytes	37	12
CF / 7 bytes	30	5
No sending, waiting for data		
CF / 7 bytes	23	23
CF / 7 bytes	16	16
CF / 7 bytes	9	9
CF / 7 bytes	2	2
CF / 2 bytes	0	0

Figure 7: Example of transmit process

1: The PduR asks for the transmission of 50 data bytes.
2: The CanTp asks the PduR for the data. The PduR provides 25 bytes of data (by conception in this example, it is not able to provide 50 bytes of data directly).
3: The CanTp starts the transmission of the payload data. After the second consecutive frame transmission, there are still 5 data bytes available in the buffer. As a consecutive frame will contain 7 data bytes, the CanTp should request new data from the PduR, in order to have enough data to send. Therefore, it should store the 5 data bytes available and afterwards request the buffer.
4: The CanTp asks the PduR for the data. The PduR provides 25 bytes of data.
5: The CanTp continues the transmission of the payload data.

This figure shows the necessity for CAN Transport Layer to use a local buffer to store some data before requesting for more data.
The new data should be provided before the N_Cs timer expires. To extend this timing constraint, the CAN Transport Layer could use a larger internal buffer and request the next data (`PduR_CanTpProvideTxBuffer()`) before the current data is transmitted (or not sufficient to transmit a complete consecutive frame).

The next figure is an example of an N-SDU receiving 49 bytes, with two buffers of 25 bytes provided.



received frame and length	Remaining bytes in the SDU	Remaining bytes in the provided buffer
FF / 6 bytes	43	19
CF / 7 bytes	36	12
CF / 7 bytes	29	5
CF / 7 bytes	22	0
No receiving, waiting for available buffer.		
CF / 7 bytes	15	16
CF / 7 bytes	8	9
CF / 7 bytes	1	2
CF / 2 bytes	0	1

Figure 8: 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 provides a buffer of 25 bytes (by conception in this example, it is not able to provide a buffer of 49 bytes directly)
3: The CanTp manages the payload data reception until the buffer is full (on the third consecutive frame). On this third consecutive frame the CAN Transport Protocol can only store 5 bytes in the buffer. Therefore, it should request a new buffer and temporarily store the remaining 2 bytes in a local buffer.
4: The CanTp asks the PduR for a new buffer in order to store the data received subsequently.
5: The CanTp copies the 2 bytes, temporarily stored in local buffer, to the buffer provided by the PduR and manages the payload data reception until the end of reception.
6: 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 buffer provided,
- amount of receiving bytes.

When the last buffer is returned to the upper layer (PduR_CanTpRxIndication()), the last bytes (in the example just the last byte) could be unused.

The upper layer shall take care identify these unused bytes with the knowledge of the total N-SDU length (function parameter of PduR_CanTpProvideRxBuffer()).

Another solution to avoid unused bytes is for the upper layer to provide the last buffer with the exact length, which should be received.

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 configured 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.

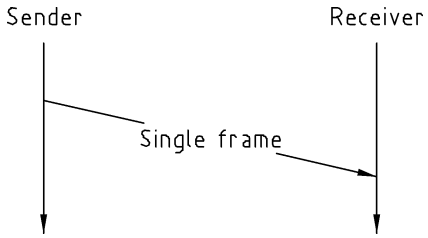


Figure 9: Example of single part message

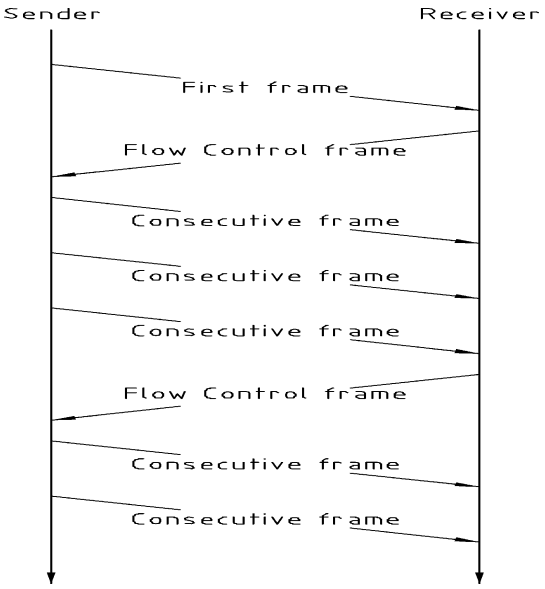


Figure 10: Example of multiple parts message

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_PARAM_CONFIG.

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.
However, if the message is configured to use an extended addressing format, the CanTp module must fill the first byte of each transmitted segment (SF, FF and CF) with the N_TA value. Therefore a CAN NSduId may also be related to a N_TA value.

FC protocol data units give receivers the possibility of controlling senders' data flow by authorizing or delaying transmission of subsequent CF N-PDUs. For extended addressing format, the first data byte of the FC also contains the N_TA value.
CanTp094: Thus the CAN LSduId of a FC frame combined with its N_TA value (e.g. the N_AI) shall only identify one CAN NSduId.

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 following figure summarizes these discussions.

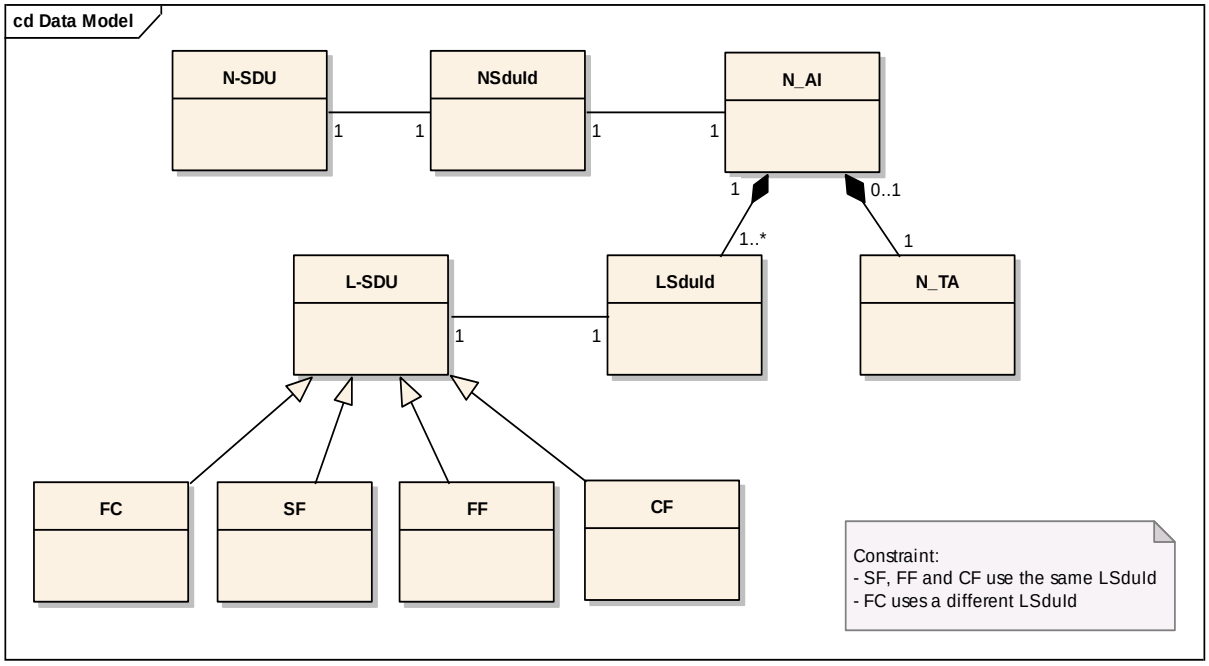


Figure 11: Possible links between NSduId and LSduId

7.3.7 Concurrent connection

In this second release document, the CAN Transport Layer will only be used for diagnosis communication (i.e. the CanTp is used to transfer DCM I-PDU). However,

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

CanTp120: It shall be possible to configure concurrent connections in the CanTp module. The connection channels are only destined for CAN TP internal use, so they are not accessible externally. 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.

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.

CanTp190: The CanTp module shall route the N-SDU through the correctly configured connection channel.

Note that this mechanism 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.

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.

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.

CanTp124: When an SF or FF is received, and the corresponding connection channel is currently receiving (state `CANTP_RX_PROCESSING`) the same connection (same N_AI), the CanTp module shall abort the reception in progress and shall process the received frame as the start of a new reception.
When an SF or FF is received for another connection (different N_AI) on an active connection channel, the SF or FF shall be ignored.

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 per N-SDU by using either CanTpRxPaddingActivation for a Rx N-SDU or CanTpTxPaddingActivation for a Tx N-SDU.

CanTp320: If CanTpRxPaddingActivation is equal to CANTP_ON for an Rx N-SDU, the CanTp module shall only accept SF Rx N-PDUs or last CF Rx N-PDUs, belonging to that N-SDU, with a length of eight bytes (i.e. PduInfoPtr. SduLength = 8).

CanTp321: If CanTpRxPaddingActivation is equal to CANTP_ON for an Rx N-SDU, and the CanTp module receives by means of `CanTp_RxIndication()` call an SF Rx N-PDU belonging to that N-SDU, with a length smaller than eight bytes (i.e. PduInfoPtr. SduLength < 8), the CanTp shall reject the reception. If the development error detection is enabled, a development error CANTP_E_PADDING shall be triggered.

CanTp322: If CanTpRxPaddingActivation is equal to CANTP_ON for an Rx N-SDU, and the CanTp module receives by means of CanTp_RxIndication() call a last CF Rx N-PDU belonging to that N-SDU, with a length smaller than eight bytes (i.e. PduInfoPtr. SduLength != 8), the CanTp shall abort the ongoing reception by calling `PduR_CanTpRxIndication()` with the result `NTFRSLT_PADDING_E_NOT_OK`. If the development error detection is enabled, a development error CANTP_E_PADDING shall be triggered.

CanTp323: If CanTpRxPaddingActivation is equal to CANTP_ON for an Rx N-SDU, the CanTp module shall transmit FC N-PDUs with a length of eight bytes. Unused bytes in N-PDU shall be updated with CANTP_PADDING_BYTE (see CanTp298).

CanTp324: If CanTpTxPaddingActivation is equal to CANTP_ON for a Tx N-SDU, the CanTp module shall transmit by means of `CanIf_Transmit()` call, SF Tx N-PDU or last CF Tx N-PDU that belongs to that Tx N-SDU with the length of eight bytes(i.e. PduInfoPtr.SduLength = 8). Unused bytes in N-PDU shall be updated with CANTP_PADDING_BYTE (see CanTp298).

CanTp325: If CanTpTxPaddingActivation is equal to CANTP_ON for a Tx N-SDU, and if a FC N-PDU is received for that Tx N-SDU on an ongoing transmission, by means of `CanTp_RxIndication()` call, and the length of this FC is smaller than eight bytes (i.e. PduInfoPtr.SduLength < 8) the CanTp module shall abort the transmission session by calling `PduR_CanTpTxConfirmation()` with the result `NTFRSLT_PADDING_E_NOT_OK`. If the development error detection is enabled, a development error CANTP_E_PADDING shall be triggered.

CanTp116: In both padding and no padding modes, the CanTp module shall only transfer used data bytes to the upper layer.

CanTp059: The CanTp module shall use the value configured in the CanTpPaddingByte configuration parameter for the padding bytes.

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.

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.

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_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)	Ignore	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 1: Handling of unexpected N-PDU arrivals

[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[3]	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).
- 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.

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

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.

CanTp101: Development error values are of type uint8.

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 TX I-Pdu On any Null-Pointer given on API calls	Development	Could be a combination of: CANTP_E_PARAM_CONFIG CANTP_E_PARAM_ID CANTP_E_PARAM_ADDRESS	0x01 0x02 0x04
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 (e.g. the Tx buffer address is inaccessible or NULL)	Development	CANTP_E_INVALID_TX_BUFFER	0x50
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/receive PDU (e.g. a transmit N-PDU has a length equal to 8)	Development	CANTP_E_PADDING	0x70
Invalid return value provided by function PduR_CanTpProvideRxBuffer()	Development	CANTP_E_INVALID_BUFREQ	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	Development	CANTP_E_OPER_NOT_SUPPORTED	0x0A
Another error occurred during a reception or a transmission	Development	CANTP_E_COM	0x0B

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.

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.

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.

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.

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]: `void Det_ReportError(ModuleId, InstanceId, ApiId, ErrorId)` to report development errors.

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.

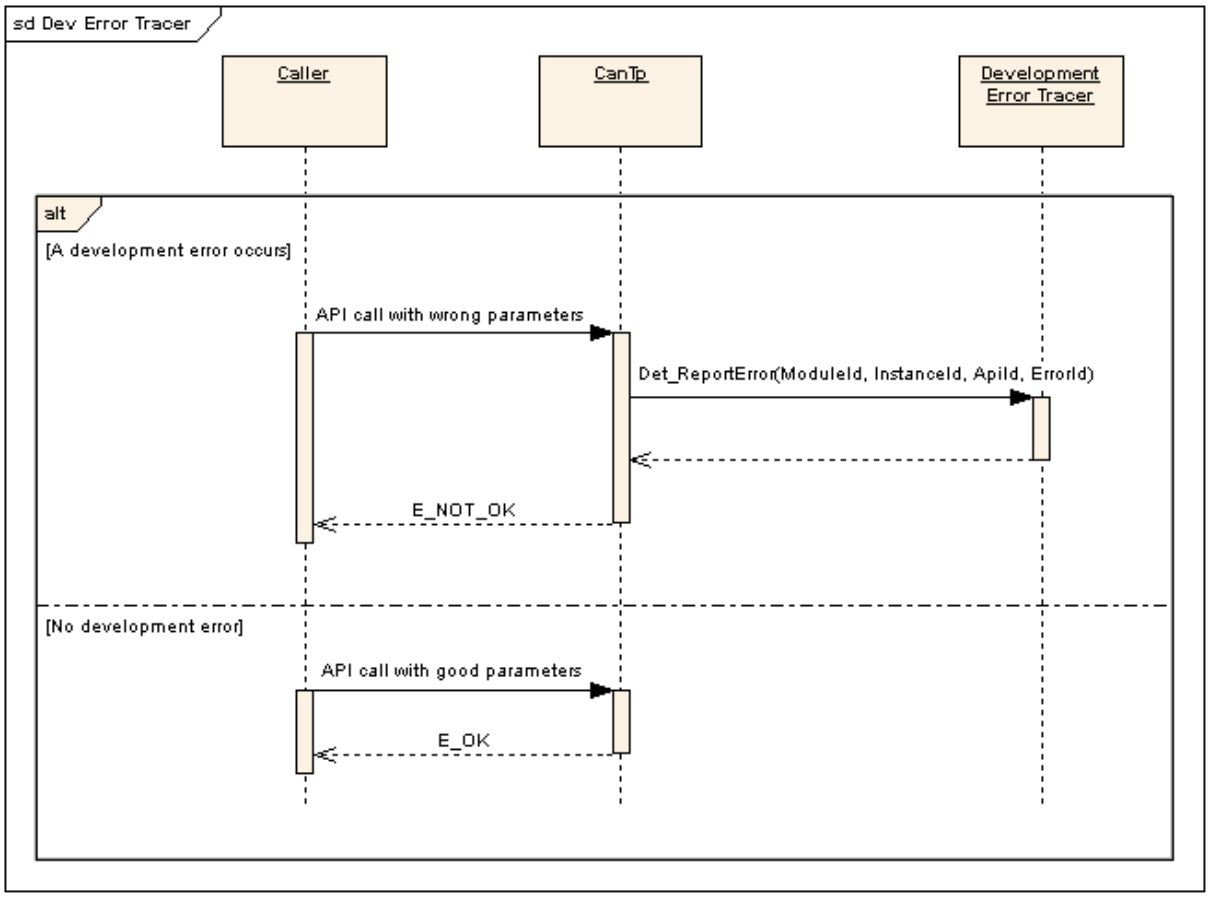


Figure 12: Development error reporting

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 (e.g. As, Bs, Cs, Ar, Br, Cr timeout), the CanTp module shall call the DET with ErrorId= CANTP_E_COM.

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
	TPParameterType
Dem	Dem_EventIdType
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.

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

None.

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

8.3.1 CanTp_Init

CanTp208:

Service name:	CanTp_Init

Syntax:	`void CanTp_Init(`
	
	`)`

Service ID[hex]:	0x01
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This function initializes the CanTp module.

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.

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:	Non 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).
-

CanTp163: The function CanTp_GetVersionInfo shall be pre compile time configurable (`On/Off`) by the configuration parameter: `CANTP_VERSION_INFO_API`.

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.

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(`
	
	`)`

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 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* PduInfoPtr )`	
Service ID[hex]:	0x03	
Sync/Async:	Synchronous	
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
	PduInfoPtr	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).

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.

Because CanTp has limited buffering capability, 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 CanTpProvideTxBuffer 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.

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_CanTpProvideTxBuffer service.

8.3.5 CanTp_CancelTransmit

CanTp246:

Service name:	CanTp_CancelTransmit	
Syntax:	`Std_ReturnType CanTp_CancelTransmit( PduIdType CanTpTxPduId )`	
Service ID[hex]:	0x08	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	CanTpTxPduId	This parameter contains the CAN TP instance unique identifier of the CAN N-SDU which transfer has to be cancelled.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Cancellation request of the transfer of the specified CAN N-SDU is accepted. E_NOT_OK: Cancellation request of the transfer of the specified CAN N-SDU is rejected, e. g. cancellation is requested at the receiver in an 1:n connection or in an unsegmented transfer at the receiver or cancellation is not allowed for the corresponding channel.
Description:	This service primitive is used to cancel the transfer of pending CAN N-SDUs. The connection is identified by CanTpTxPduId. When the function returns, no transmission is in progress anymore with the given N-SDU identifier. This function has to be called with the PDU-Id of the CanTp, i.e. the upper layer has the same PDU-Id as for the FrTp_Transmit() call.	

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

CanTp308: If development error detection is enabled the function CanTp_CancelTransmit shall check the validity of CanTpTxPduId parameter. If the parameter value is invalid (received N-SDU, out of range), the CanTp_CancelTransmit function shall raise the development error CANTP_E_PARAM_ID and return E_NOT_OK.

8.3.6 CanTp_CancelReceive

CanTp310:

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	None	

(inout):		
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. When the function returns, no reception is in progress anymore with the given N-SDU identifier.	

The service CanTp_CancelReceive cancels the reception of an N-SDU initiated by the reception of a First Frame.

CanTp311: 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.

CanTp312: 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.

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).

CanTp305: 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.

CanTp306: 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.

CanTp307: 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.

8.3.8 CanTp_ReadParameter

CanTp303:

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 is stored.
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.	

The service CanTp_ReadParameter is used to read the current value of the reception parameters BS and STmin for a specific received N-SDU.

CanTp304: If development error detection is enabled the function CanTp_ReadParameter shall check the validity of function parameters (Identifier, Parameter and pointer to value). If any of the parameter value is invalid (Tx N-SDU, undefined parameter, NULL pointer) the CanTp_ReadParameter function shall raise the development error CANTP_E_PARAM_ID and returns E_NOT_OK.

8.3.9 Main Function

CanTp213:

Service name:	CanTp_MainFunction

Syntax:	`void CanTp_MainFunction(`
	
	`)`

Service ID[hex]:	0x06
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
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.

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 CanTpRxPduId,`
	`  const PduInfoType* PduInfoPtr`
	`)`	

Service ID[hex]:	0x04	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	CanTpRxPduId	ID of CAN L-PDU that has been received. Identifies the data that has been received. Range: 0..(maximum number of L-PDU IDs received) - 1
	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:	This function is called by the CAN Interface after a successful reception of a Rx CAN L-PDU.	

CanTp019: 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).

CanTp234: The function CanTp_RxIndication shall be implemented using a pre-compile macro

8.4.2 CanTp_TxConfirmation

CanTp215:

Service name:	CanTp_TxConfirmation	

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

Service ID[hex]:	0x05	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	CanTpTxPduId	ID of CAN L-PDU that has been transmitted. Range: 0..(maximum number of L-PDU IDs received) - 1
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	All transmitted CAN frames belonging to the CAN Transport Layer will be confirmed by this function.	

CanTp020: 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).

CanTp237: The function CanTp_TxConfirmation shall be implemented using a pre-compile macro

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	CANIF005: 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. The corresponding CAN controller and HTH have to be resolved by the CanTxPduId. A transmit request has not been accepted, if the controller mode is not STARTED and/or the channel mode at least for the transmit path is not online or offline active. One call of this function results in one call of Can_Write(Hth, *PduInfo). Development errors: Invalid values of CanTxPduId or PduInfoPtr will be reported to the development error tracer (CANIF_E_INVALID_TXPDUID or CANIF_E_PARAM_POINTER). If the CAN Interface was not initialized before, the call of this function will be reported to the development error tracer (CANIF_E_UNINIT). The function returns with E_NOT_OK. Caveats: During the call of this API the buffer of PduInfoPtr is controlled by the CAN Interface may not be accessed for read/write from another call context. After return of this call the ownership changes to the upper layer. The CAN Interface must be initialized after Power ON.
Dem_ReportErrorStatus	Reports errors to the DEM.
PduR_CanTpProvideRxBuffer	Provides Rx buffer for the CAN TP.
PduR_CanTpProvideTxBuffer	Provide Tx data for the CAN TP.
PduR_CanTpRxIndication	Rx indicator for the CAN TP
PduR_CanTpTxConfirmation	Tx confirmation for the CAN TP

Note: As CanTp modules does not define any production errors the Dem_ReportErrorStatus is not a mandatory interface; however it might be included for build compatibility reasons or for future possibility of production error definition.

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

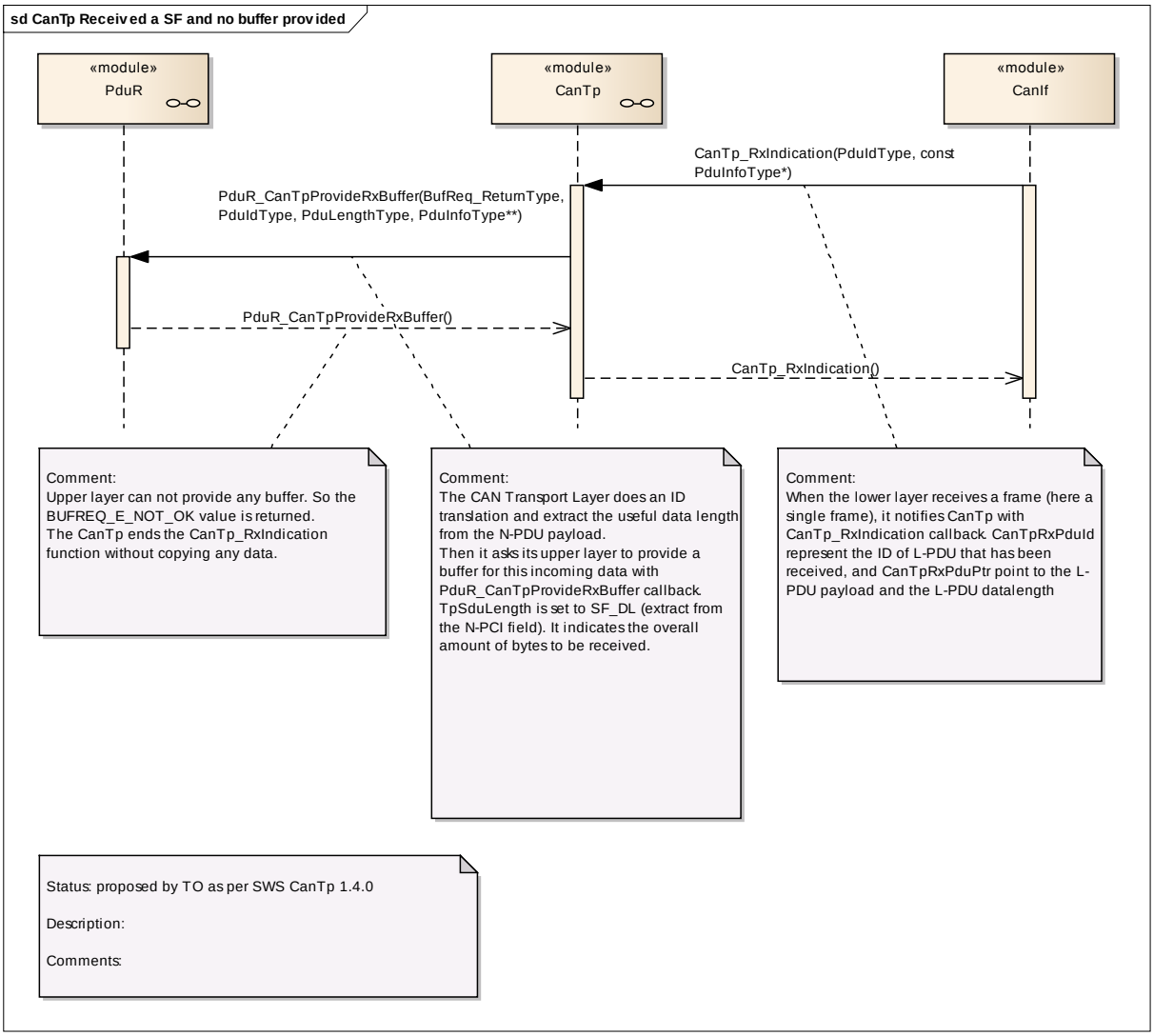
The goal of this chapter is to make it easier to understand the CAN Transport Layer by describing most of the more frequent and complicated use cases. Thus, the following diagram sequences are not exhaustive and do not reflect all the specified API possibilities.

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 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

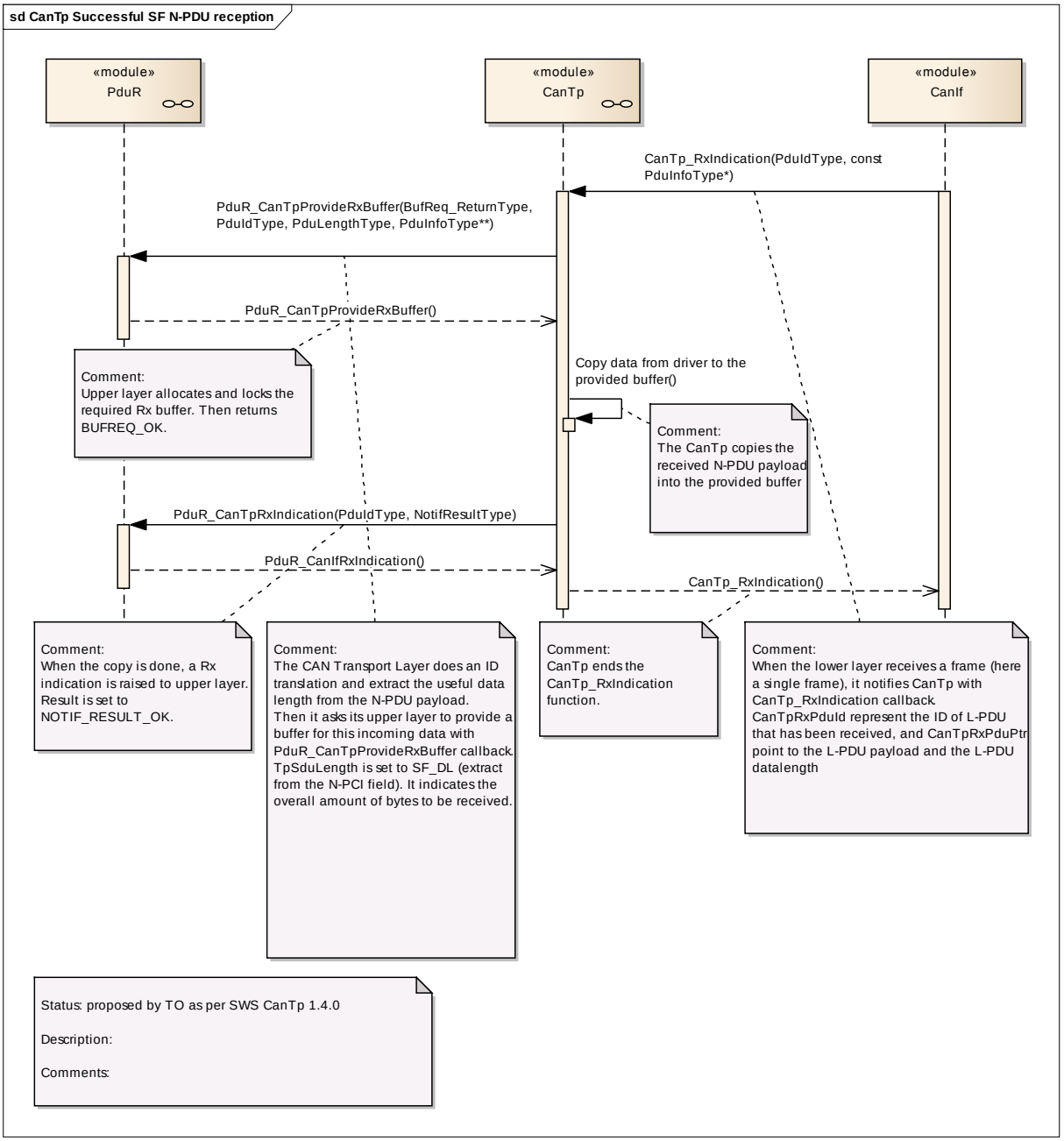
Transition	*Name*	*Description*
1	`CanTp_RxIndication ( CanTpRxPduId, CanTpRxPduPtr )`	When the lower layer receives a frame (here a single frame), it notifies CanTp by means of a CanTp_RxIndication callback. CanTpRxPduId represents the ID of L-PDU that has been received, and CanTpRxPduPtr indicates the L-PDU payload and the L-PDU datalength
2	`PduR_CanTpProvideRxBuffer( 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_CanTpProvideRxBuffer 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.

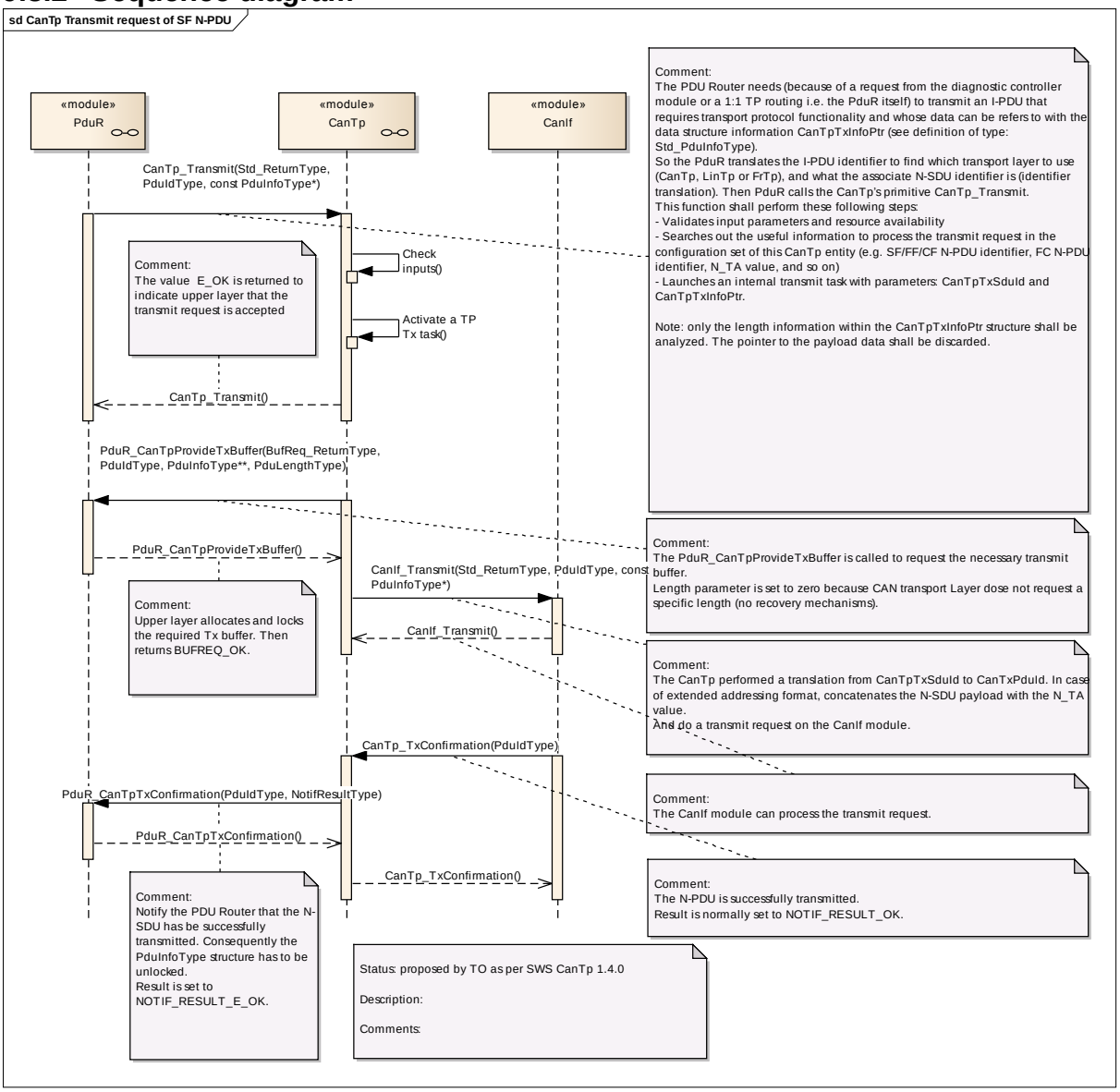
Transition	Name	Description
2	PduR_CanTpProvideRxBuffer (CanTpRxSduId, TpSduLength, PduInfoPtr)	The CAN Transport Layer performs an ID translation and extract the useful data length from the N-PDU payload. Then it asks its upper layer to provide a buffer for this incoming data with a PduR_CanTpProvideRxBuffer 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		The CanTp 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

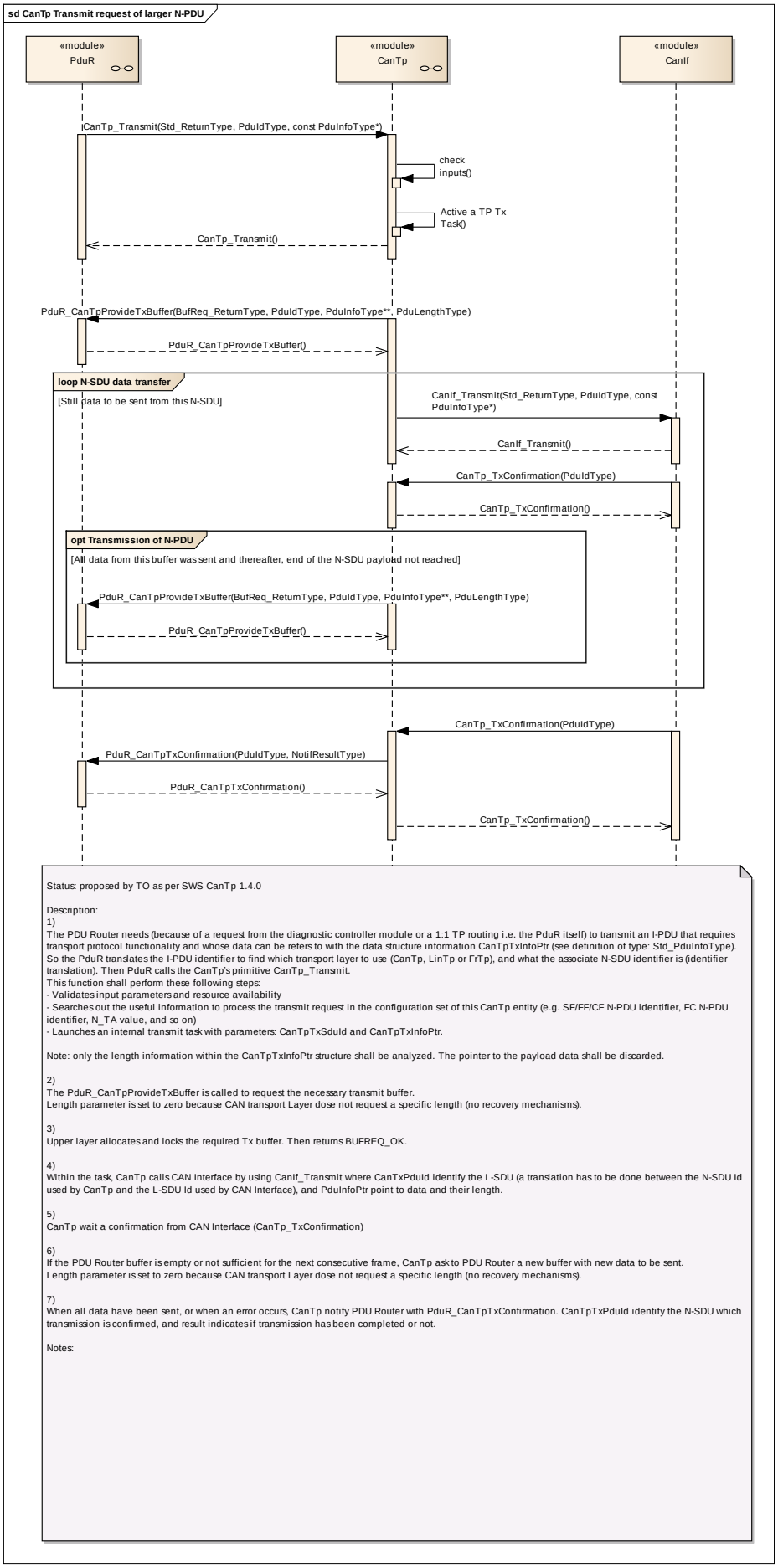
Transition	Name	Description
1	`CanTp_Transmit( CanTpTxSduId, CanTpTxInfoPtr )`	The PDU Router needs (because of a request from the diagnostic controller module or a 1:1 TP routing – i.e. the PduR itself) to transmit an I-PDU that requires transport protocol functionality and whose data can be refered to with the data structure information `CanTpTxInfoPtr` (see definition of type: `Std_PduInfoType`). Thus, the PduR translates the I-PDU identifier to establish 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 will perform the following steps: - Validates input parameters and resource availability - Searches for 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 the parameters: `CanTpTxSduId` and `CanTpTxInfoPtr`. Note: only information concerning length, within the CanTpTxInfoPtr structure, will be analyzed. The payload indicator data should 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_CanTpProvideTxBuffer ( CanTpTxSduId, PduInfoPtr, Length=0 )`	The `PduR_CanTpProvideTxBuffer` is called upon to request the necessary transmit buffer. `Length` parameter is set to zero because the CAN transport Layer does not request a specific length (no recovery mechanism).
4	`BUFREQ_OK`	Upper layer allocates 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 addressing format)
- The transmission is successfully processed

9.4.2 Sequence diagram


sd CanTp Transmit request of larger N-PDU
«module» PduR
«module» CanTp
«module» CanIf
CanTp_Transmit(Std_ReturnType, PduIdType, const PduInfoType*)
check inputs()
Active a TP Tx Task()
CanTp_Transmit()
PduR_CanTpProvideTxBuffer(BufReq_ReturnType, PduIdType, PduInfoType**, PduLengthType)
PduR_CanTpProvideTxBuffer()
loop N-SDU data transfer
[Still data to be sent from this N-SDU]
CanIf_Transmit(Std_ReturnType, PduIdType, const PduInfoType*)
CanIf_Transmit()
CanTp_TxConfirmation(PduIdType)
CanTp_TxConfirmation()
opt Transmission of N-PDU
[All data from this buffer was sent and thereafter, end of the N-SDU payload not reached]
PduR_CanTpProvideTxBuffer(BufReq_ReturnType, PduIdType, PduInfoType**, PduLengthType)
PduR_CanTpProvideTxBuffer()
CanTp_TxConfirmation(PduIdType)
PduR_CanTpTxConfirmation(PduIdType, NotifResultType)
PduR_CanTpTxConfirmation()
CanTp_TxConfirmation()
Status: proposed by TO as per SWS CanTp 1.4.0
Description:
1)
The PDU Router needs (because of a request from the diagnostic controller module or a 1:1 TP routing i.e. the PduR itself) 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)
The PduR_CanTpProvideTxBuffer is called to request the necessary transmit buffer.
Length parameter is set to zero because CAN transport Layer dose not request a specific length (no recovery mechanisms).
3)
Upper layer allocates and locks the required Tx buffer. Then returns BUFREQ_OK.
4)
Within the task, CanTp calls CAN Interface by using CanIf_Transmit where CanTxPduId identify the L-SDU (a translation has to be done between the N-SDU Id used by CanTp and the L-SDU Id used by CAN Interface), and PduInfoPtr point to data and their length.
5)
CanTp wait a confirmation from CAN Interface (CanTp_TxConfirmation)
6)
If the PDU Router buffer is empty or not sufficient for the next consecutive frame, CanTp ask to PDU Router a new buffer with new data to be sent.
Length parameter is set to zero because CAN transport Layer dose not request a specific length (no recovery mechanisms).
7)
When all data have been sent, or when an error occurs, CanTp notify PDU Router with PduR_CanTpTxConfirmation. CanTpTxPduId identify the N-SDU which transmission is confirmed, and result indicates if transmission has been completed or not.
Notes:

9.4.3 Transition description

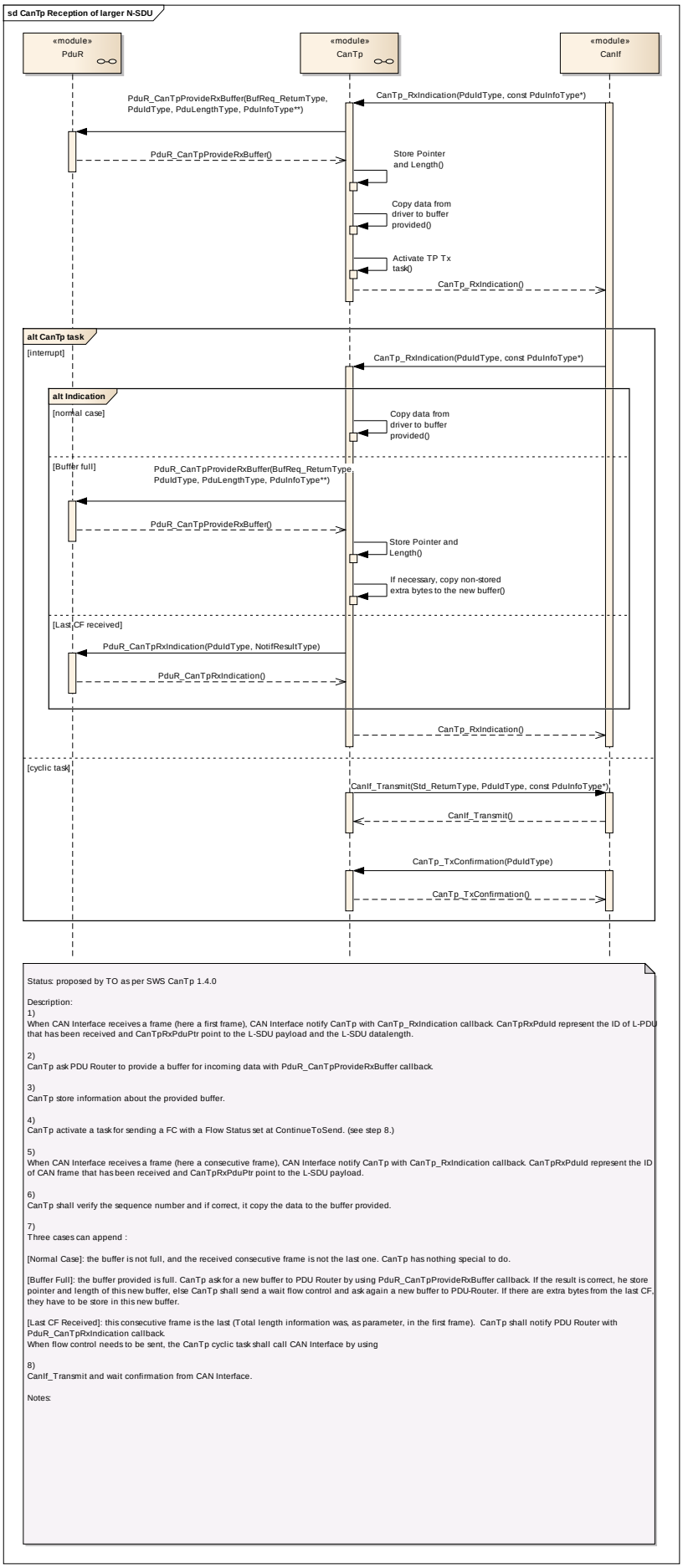
Transition	Name	Description
1	CanTp_Transmit (CanTpTxSduId, CanTpTxInfoPtr)	The PDU Router needs (because of a request from the diagnostic controller module or a 1:1 TP routing – i.e. the PduR itself) to transmit an I-PDU that requires transport protocol functionality and whose data can be referred to with the data structure information CanTpTxInfoPtr (see definition of type: Std_PduInfoType). Thus, the PduR translates the I-PDU identifier to establish 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 should perform the following steps: - Validate input parameters and resource availability - Search for 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) - Launch an internal transmit task with parameters: CanTpTxSduId and CanTpTxInfoPtr. Note: only information concerning length within the CanTpTxInfoPtr structure will be analyzed. The indicactor to the payload data should be discarded.
2	PduR_CanTpProvideTx Buffer (CanTpTxSduId, PduInfoPtr, Length=0)	The PduR_CanTpProvideTxBuffer is called upon to request the necessary transmit buffer. Length parameter is set to zero because the CAN transport Layer does not request a specific length (no recovery mechanism).
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_TxConfirmatio n(CanTpTxPduId,)	CanTp awaits a confirmation from the CAN Interface (CanTp_TxConfirmation)
6	PduR_CanTpProvideTx Buffer (CanTpTxTxSduId, PduInfoPtr, Length=0)	If the PDU Router buffer is empty or not sufficient for the next consecutive frame, CanTp asks the PDU Router for a new buffer, with new data, to be sent. Length parameter is set to zero because the CAN transport Layer does not request a specific length (no recovery mechanism).
7	PduR_CanTpTxConfirm ation (CanTpTxSduId, Result)	When all data has been sent, or when an error occurs, CanTp notifies the PDU Router by means of PduR_CanTpTxConfirmation. CanTpTxPduId informs the N-SDU which transmission has been confirmed, and result indicates whether the 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



Status: proposed by TO as per SWS CanTp 1.4.0

Description:
1)
When CAN Interface receives a frame (here a first frame), CAN Interface notify CanTp with CanTp_RxIndication callback. CanTpRxPduId represent the ID of L-PDU that has been received and CanTpRxPduPtr point to the L-SDU payload and the L-SDU datalength.

2)
CanTp ask PDU Router to provide a buffer for incoming data with PduR_CanTpProvideRxBuffer callback.

3)
CanTp store information about the provided buffer.

4)
CanTp activate a task for sending a FC with a Flow Status set at ContinueToSend. (see step 8.)

5)
When CAN Interface receives a frame (here a consecutive frame), CAN Interface notify CanTp with CanTp_RxIndication callback. CanTpRxPduId represent the ID of CAN frame that has been received and CanTpRxPduPtr point to the L-SDU payload.

6)
CanTp shall verify the sequence number and if correct, it copy the data to the buffer provided.

7)
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 ask for a new buffer to PDU Router by using PduR_CanTpProvideRxBuffer callback. If the result is correct, he store pointer and length of this new buffer, else 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 store 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.
When flow control needs to be sent, the CanTp cyclic task shall call CAN Interface by using

8)
CanIf_Transmit and wait confirmation from CAN Interface.

Notes:

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_CanTpProvideRxBuffer ( CanTpRxSduId, TpSduLength, PduInfoPtr )`	CanTp asks the PDU Router to provide a buffer for incoming data by means of a `PduR_CanTpProvideRxBuffer` callback.
3		CanTp stores information about the buffer provided.
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 will verify the sequence number and if correct, copy the data to the buffer provided.

7	`Nothing`
	`or`
	`PduR_CanTpProvideRxBuffer ( CanTpRxSduId, TpSduLength, PduInfoPtr )`
	`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 asks the PDU Router for a new buffer by means of a `PduR_CanTpProvideRxBuffer` callback. If the result is correct, it stores the indication and length of this new buffer. Otherwise CanTp sends a wait flow control and asks the PDU-Router once again for a new buffer. 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 is known using the FF_DL parameter in the first frame). CanTp notifies PDU Router by means of a `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 await 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.

CanTp147: The consistency of the configuration must be checked by the configuration tool at configuration time. 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 1: only pre-compile time configuration parameters, variant 2: mix of pre-compile- and post build time-configuration parameters. In one variant, a parameter can only be of one configuration class.

10.1.3 Containers

Containers structure the set of configuration parameters. This means:

- *all* configuration parameters are kept in containers.
- (sub-) containers can reference (sub-) containers. It is possible to assign a multiplicity to these references. 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[4]	Refer here to (a) variant(s).
	Link time	see[5]	Refer here to (a) variant(s).
	Post Build	see[6]	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.		

[4] see the explanation below this table - Pre-compile time
[5] see the explanation below this table - Link time
[6] 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 1: all parameters are configured at compile time.
Variant 2: some parameters are configured at compile time, some parameters are configured at post build time.

10.2.2 CanTp

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

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

10.2.3 CanTpGeneral

SWS Item	**CanTp238 :**
Container Name	CanTpGeneral{CanTpConfiguration}
Description	This container contains the general configuration parameters of the CanTp module.
Configuration Parameters	

SWS Item	**CanTp299 :**		
Name	CanTpChangeParameterApi {CANTP_CHANGE_PARAMETER_API}		
Description	This parameter, if set to true, enables the CanTp_ChangeParameter Api for this Module.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**CanTp239 :**		
Name	CanTpDevErrorDetect {CANTP_DEV_ERROR_DETECT}		
Description	Switches the Development Error Detection and Notification ON or OFF		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build	--	

	time		
Scope / Dependency	scope: Module		

SWS Item	**CanTp240 :**		
Name	CanTpMainFunctionPeriod {CANTP_MAIN_FUNCTION_PERIOD}		
Description	Allow to configure the time for the MainFunction (as float in seconds). Please note: This configuration value shall be equal to the value in the ScheduleManger module.		
Multiplicity	1		
Type	FloatParamDef		
Range	0 .. 0.255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	L	VARIANT-POST-BUILD
Scope / Dependency	scope: ECU		

SWS Item	**CanTp298 :**		
Name	CanTpPaddingByte {CANTP_PADDING_BYTE}		
Description	Used for the initialization of unused bytes with a certain value		
Multiplicity	1		
Type	IntegerParamDef		
Range	0 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**CanTp282 :**		
Name	CanTpTc {CANTP_TC}		
Description	Preprocessor switch for enabling Transmit Cancellation and Receive Cancellation.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

No Included Containers

10.2.4 CanTpChannel

SWS Item	**CanTp288 :**
Container Name	CanTpChannel
Description	This container contains the configuration parameters of the CanTp channel.
Configuration Parameters	

SWS Item	**CanTp289 :**
Name	CanTpChannelMode
Description	The CAN Transport Layer supports half and full duplex channel modes.
Multiplicity	1

Type	EnumerationParamDef		
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.

10.2.5 CanTpRxNSdu

SWS Item	**CanTp137 :**
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	**CanTp242 :**		
Name	CanTpAddressingFormat {CANTP_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.		
Multiplicity	1		
Type	EnumerationParamDef		
Range	CANTP_EXTENDED	Extended 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	**CanTp243 :**		
Name	CanTpBs {CANTP_BS}		
Description	Sets the maximum 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. Note: For reasons of buffer length, the CAN Transport Layer can adapt the BS value within the limit of this maximum BS		
Multiplicity	1		
Type	IntegerParamDef		
Range	..		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	L	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**CanTp244 :**
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	1		
Type	FloatParamDef		
Range	-INF .. INF		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	L	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**CanTp245 :**		
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	FloatParamDef		
Range	-INF .. INF		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	L	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**CanTp246 :**		
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	1		
Type	FloatParamDef		
Range	-INF .. INF		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	L	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**CanTp248 :**		
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. This parameter is set to obsolete and will be removed in future.		
Multiplicity	0..1		
Type	IntegerParamDef		
Range	..		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**CanTp276 :**
Name	CanTpRxNSduId {CANTP_RXNSDU_ID}
Description	Unique identifier to a structure that contains all useful information to process the

	reception of a RxNsdu.		
Multiplicity	1		
Type	IntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 ..		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**CanTp249 :**		
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	EnumerationParamDef		
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 :**		
Name	CanTpRxTaType {CANTP_TA_TYPE}		
Description	Declares the communication type of this Rx N-SDU.		
Multiplicity	1		
Type	EnumerationParamDef		
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 :**		
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	1		
Type	IntegerParamDef		
Range	..		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	L	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**CanTp252 :**
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	1		
Type	IntegerParamDef		
Range	..		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	L	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**CanTp300 :**		
Name	CanTpStaticBlockSize {CANTP_STATIC_BLOCK_SIZE}		
Description	If this parameter is TRUE, the BlockSize will remain the same as in the first flow control. If this parameter is FALSE, the BlockSize is dynamic.		
Multiplicity	1		
Type	BooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**CanTp241 :**		
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	X	VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency			

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

10.2.6 CanTpRxNPdu

SWS Item	**CanTp256 :**
Container Name	CanTpRxNPdu
Description	Used for grouping of the ID of a PDU and the Reference to a PDU.
Configuration Parameters	

SWS Item	**CanTp258 :**		
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 addressing format, the same N-PDU identifier can be used for several N-SDU identifiers. The distinction is made by the N_TA value (first data byte of SF or FF frames).		
Multiplicity	1		
Type	IntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 ..		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**CanTp257 :**		
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	L	VARIANT-POST-BUILD
Scope / Dependency			

No Included Containers

10.2.7 CanTpTxFcNPdu

SWS Item	**CanTp259 :**
Container Name	CanTpTxFcNPdu
Description	Used for grouping of the ID of a PDU and the Reference to a PDU.
Configuration Parameters	

SWS Item	**CanTp260 :**		
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	L	VARIANT-POST-BUILD
Scope / Dependency			

No Included Containers

10.2.8 CanTpTxNSdu

SWS Item	**CanTp138 :**
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	**CanTp262 :**		
Name	CanTpAddressingMode {CANTP_ADDRESSING_MODE}		
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).		
Multiplicity	1		
Type	EnumerationParamDef		
Range	CANTP_EXTENDED	Extended 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	**CanTp263 :**		
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	FloatParamDef		
Range	-INF .. INF		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	L	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**CanTp264 :**		
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	1		
Type	FloatParamDef		
Range	-INF .. INF		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	L	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**CanTp265 :**		
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	1		
Type	FloatParamDef		
Range	-INF .. INF		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	--	
	Post-build time	L	VARIANT-POST-BUILD
Scope / Dependency	scope: Module		

SWS Item	**CanTp267 :**		
Name	CanTpTxDl {CANTP_DL}		
Description	Data Length Code of this TxNsdu. In case of variable length message, this value indicates the minimum data length. This parameter is set to obsolete and will be removed in future.		
Multiplicity	0..1		
Type	IntegerParamDef		
Range	..		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**CanTp268 :**		
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	IntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 ..		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**CanTp269 :**		
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	EnumerationParamDef		
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 :**		
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	EnumerationParamDef		
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 :**		
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	X	VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency			

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

10.2.9 CanTpTxNPdu

SWS Item	**CanTp274 :**
Container Name	CanTpTxNPdu
Description	Used for grouping of the ID of a PDU and the Reference to a PDU.
Configuration Parameters	

SWS Item	**CanTp275 :**		
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

10.2.10 CanTpRxFcNPdu

SWS Item	**CanTp271 :**
Container Name	CanTpRxFcNPdu
Description	Used for grouping of the ID of a PDU and the Reference to a PDU.
Configuration Parameters	

SWS Item	**CanTp273 :**
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	IntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 ..		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**CanTp272 :**		
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	L	VARIANT-POST-BUILD
Scope / Dependency			

No Included Containers

10.2.11 CanTpNTa

SWS Item	**CanTp139 :**
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 :**		
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	IntegerParamDef		
Range	..		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

No Included Containers

10.2.12 CanTpNSa

SWS Item	**CanTp253 :**
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 :**
Name	CanTpNSa {CANTP_NSA}
Description	If an RxNSdu ora TxNSdu is configured for extended addressing format, this

	parameter contains the transport protocol source address's value.		
Multiplicity	1		
Type	IntegerParamDef		
Range	..		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.3 Published Information

Published information contains data defined by the implementer of the SW module that does not change when the module is adapted (i.e. configured) to the actual HW/SW environment. It thus contains version and manufacturer information.

The standard common published information like

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

is provided in the BSW Module Description Template (see [12] Figure 4.1 and Figure 7.1).

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