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

Document Version	2.2.0
Document Status	Final
Part of Release	2.1
Revision	20

Document Change History

Date	Version	Changed by	Change Description
31.05.2010	2.2.0	AUTOSAR Administration	• Adaptation to ISO15765-2 requirements • Legal disclaimer revised
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 CANTP_WFTMAX of container TxNsdu • 1 parameter added for the call of Det_ReportError • Add header files inclusions • Addition of N_Sa 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.

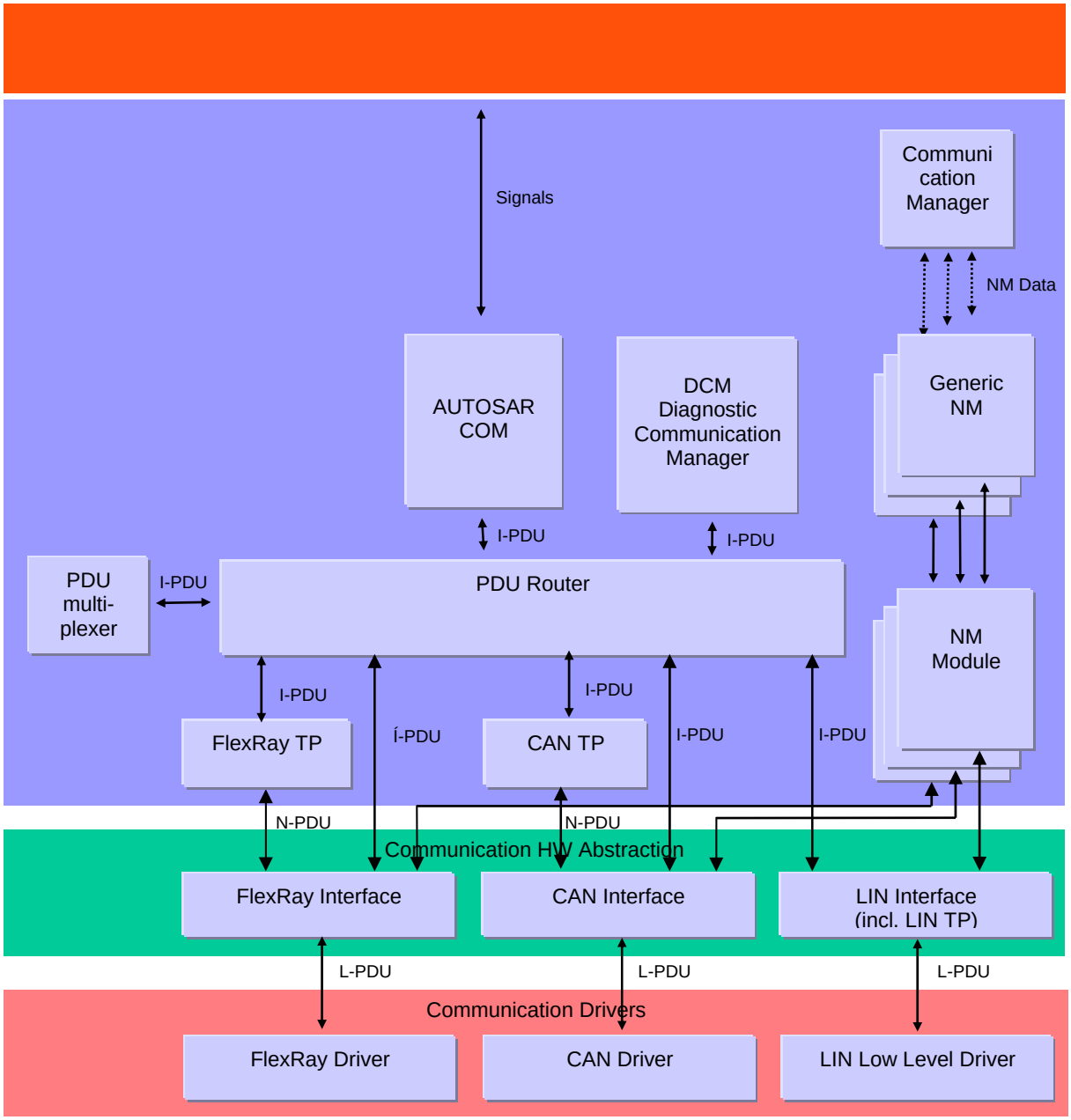


Figure 1 : AUTOSAR Communication Stack

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.

It is an AUTOSAR decision to base basic software module specification 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 [12] and ISO 15765-4 for OBD diagnostics [14]. 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 [12] 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 [12] used to encode the communication model) has the value *physical*. This means the N-SDU is used in 1 to 1 communication, thus physical addressing will be supported for all types of network layer messages. In terms of application, physical addressing is used by the external (or internal) tester if it knows the physical address of an ECU that should respond to a service request. When physical addressing is used, a point to point communication takes place (1 to 1 communication). Use cases are (for example) messages, such as “ReadDataByIdentifier” or “InputOutputControlByIdentifier”
Single connection	The CAN Transport Layer will only manage one transport protocol communication session at a time.

3 Related documentation

3.1 Input documents

[1] List of Basic Software Modules,
AUTOSAR_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

3.2 Related standards and norms

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

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

[14] 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,
- No transmission / reception cancellation.

Furthermore, to significantly reduce resources and runtime this CAN Transport Layer implementation does not support full-duplex communication.

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.

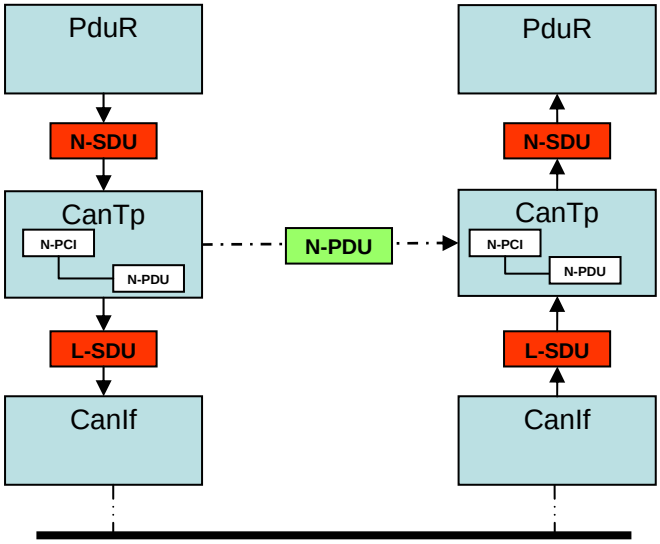


Figure 2: CAN Transport Layer interactions

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
- Minimum length of the N-SDU

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

5.1.6 PDU Router services

The CAN Transport Layer declares and requests certain callback functions to confirm transmission 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

General definitions will be defined in `CanTp.h` and specific definitions in `CanTp_Types.h`.

CanTp001: Constant and customizable data for module configuration at pre-compile time will be defined in `CanTp_Cfg.h`.

CanTp014: 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.

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.

CanTp156: The include file structure will be constructed as shown in Figure 3.

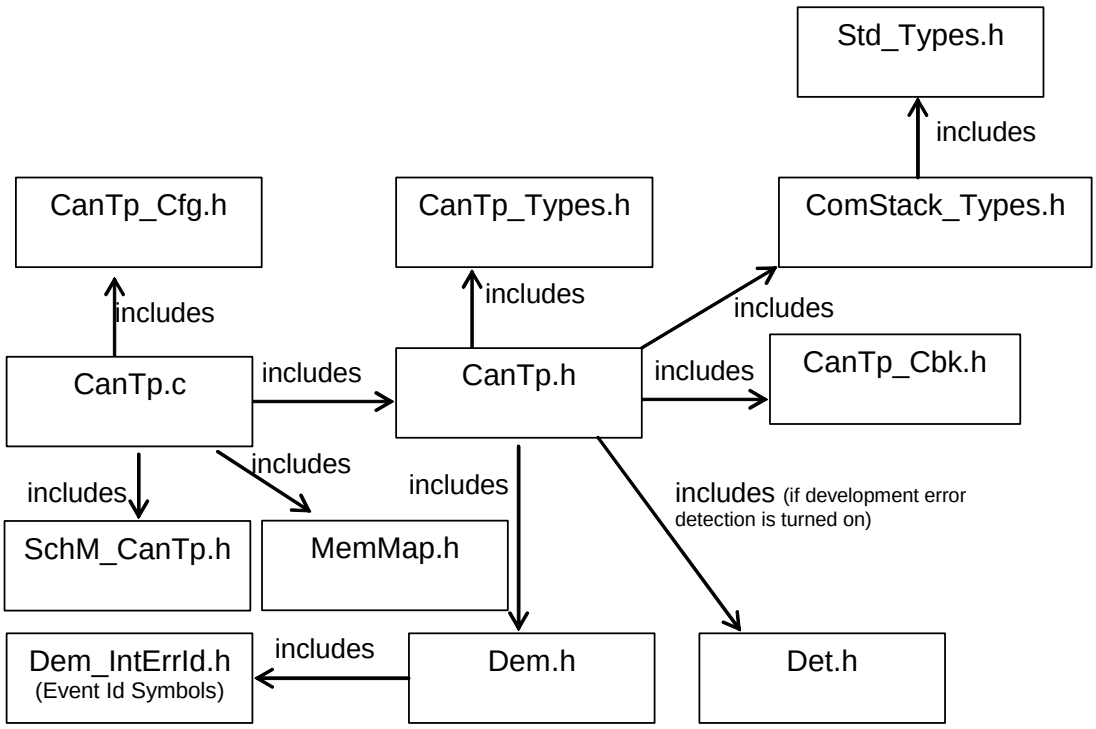


Figure 3: File Structure

- `CanTp.h` includes `CanTp_Types.h`
- `CanTp.c` includes `CanTp.h`
- `CanTp.h` includes `CanTp_Cfg.h`

CanTp022: This structure 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.
CanTp024: Each header and C file of the CAN Transport Layer will provide the possibility of version identification of the CAN Transport module by CANTP_MAJOR_VERSION, CANTP_MINOR_VERSION and CANTP_PATCH_VERSION (See chapter 10.2.6).
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.

CanTp130: The module will include the Dem.h file. As a result, the APIs required in order to report errors, as well as the necessary Event Id symbols, are included. This specification defines the names of the Event Id symbols, which are provided to the DEM configuration tool by means of XML.

CanTp157: The file CanTp.h only contains 'external' declarations of constants, global data, type definitions and services that are specified in the CAN Transport Protocol SWS.
Global data types and functions that are only used internally by the CAN Transport Protocol, are given in CanTp.c

CanTp165: BSW scheduler information is included via SchM_CanTp.h.

5.2.3 Design Rules

CanTp150: The code of the CAN Transport Protocol (as long as it is written in C) must conform to the HIS subset of the MISRA C Standard.
CanTp151: Direct use of compiler and platform specific keywords should be avoided

CanTp152: All global data with read-only properties will be indicated by explicitly assigning the keyword `const`.

CanTp153: The use of macros (instead of functions) where source code is used and runtime is critical, is allowed.

CanTp155: No global data should be defined in the header files. If global variables have to be used, the definition should take place in the C file

CanTp158: The source code of the CAN Transport Protocol module must 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 CanTp022 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	CanTp160
[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	CanTp018
[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	CanTp012 CanTp101
[BSW00338] Detection and Reporting of development errors	CanTp012 CanTp013
[BSW00369] Do not return development error codes via API	CanTp021
[BSW00339] Reporting of production relevant errors and exceptions	CanTp012 CanTp008 CanTp100
[BSW00421] Reporting of production relevant error events	CanTp100
[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	CanTp016
[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	CanTp018
[BSW00414] Parameter of init function	CanTp018
[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	CanTp032 CanTp033
[BSW01065] Usage of ISO 15765-4 specifications	CanTp032 CanTp119
[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	CanTp040 CanTp098 CanTp116
[BSW01074] Transport connection properties	CanTp137 CanTp138
[BSW01075] CAN Transport Layer Initialization	CanTp029 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 padding 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
[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.

CanTp034: The CanTp should offer 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.

CanTp032: While reading this document, it is necessary to bear in mind, that this module will follow the recommendations ISO 15765-2 (OEM enhanced diagnostics [12]) and should be able to fulfill ISO 15765-4 (Requirements for emissions-related systems [14]).

CanTp033: Therefore, if a recommendation of this ISO specification is not explicitly excluded, the AUTOSAR CAN Transport Layer implementation should 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 [12].

CanTp119: 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 CAN Transport Layer 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 CAN Transport Layer will have two internal states, `CANTP_OFF` and `CANTP_ON`.

CanTp028: After power up, the CAN Transport Layer will be in the `CANTP_OFF` state. In this state, the CanTp configuration may be updated.

CanTp029: The CanTp will change to the internal state `CANTP_ON` when the CanTp has been successfully initialized with `CanTp_Init()`. Segmentation and reassembly tasks may only be performed when the CanTp is in the `CANTP_ON` state.

CanTp030: This `CanTp_Init()` service will 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`).

CanTp031: The `CanTp_Init()` service should be called by the COM Manager before using the CanTp functionalities. Otherwise, a development error will be generated when the PDU Router or CAN Interface tries to use another service (except the CanTp_GetVersionInfo interface).

CanTp111: If the `CanTp_Init()` service is called when the CanTp is in the global state `CANTP_ON`, all current connections are lost and the module returns to state Idle.

CanTp010: A shutdown function, `CanTp_Shutdown()`, is also provided to stop the CanTp module properly. It will be called upon by the COM Manager.

The following figure summarizes all of the above requirements:

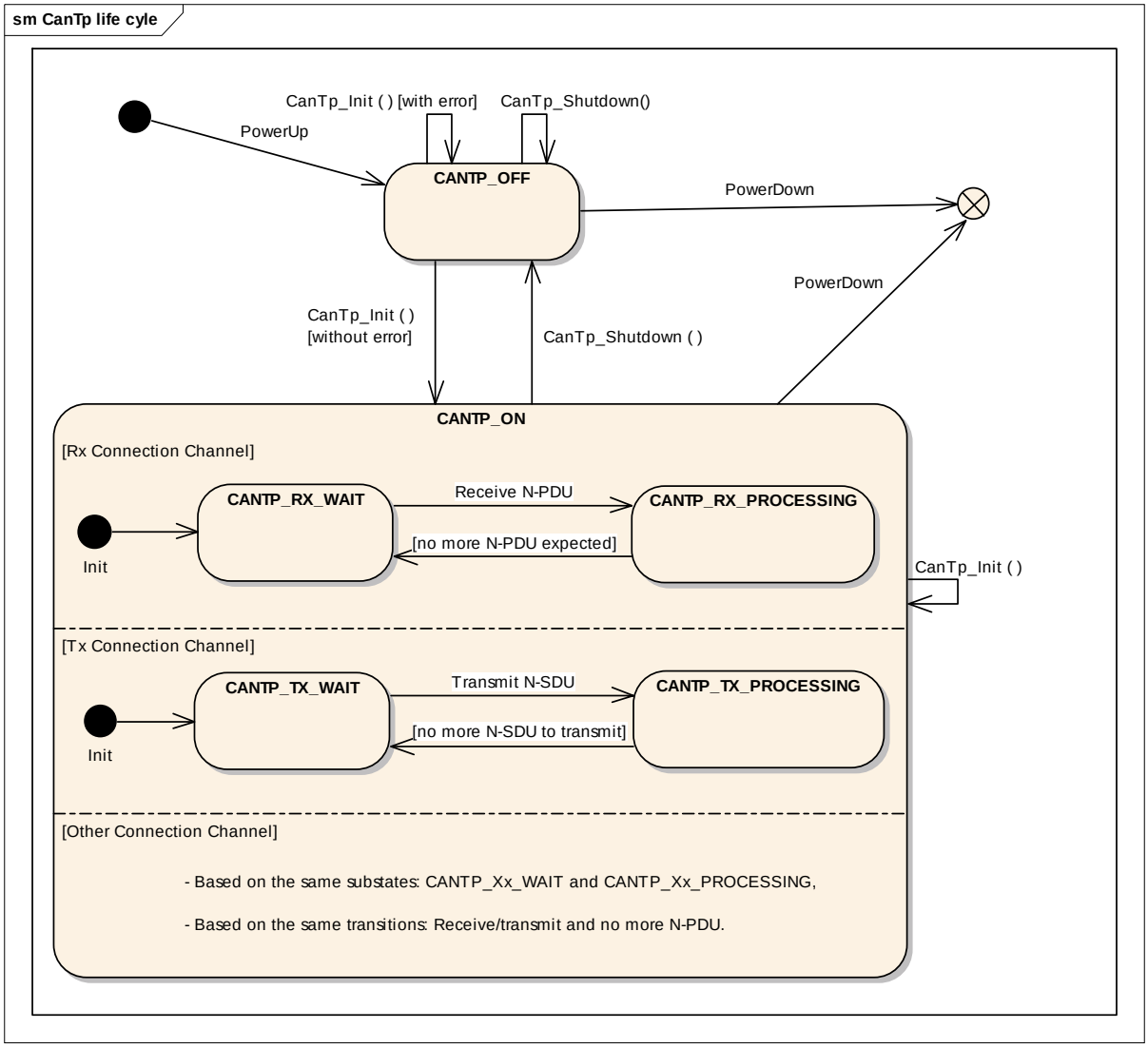


Figure 4: CAN Transport Layer life cycle

7.1.2 Transmit request

CanTp070: 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).

CanTp071: This function should be asynchronous. When the N-SDU transfer is fully processed (successfully or not) the CanTp will notify its upper layer.

CanTp072: If there is no hardware resource available, the transmit request will be rejected. In this case, CanTp_Transmit() will return the status value E_NOT_OK.

7.2 Services provided to the lower layer

CanTp073: According to the AUTOSAR specification of the communication stack, the CAN Transport Layer should provide the following two callback functions to the Can interface: `CanTp_TxConfirmation()` and `CanTp_RxIndication()`.

7.2.1 Transmit confirmation

CanTp074: The transmit confirmation function will be called by the CAN Interface module 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 CAN Transport Layer will act as if it had received an unsuccessful transmission confirmation and any late confirmation will be ignored. The CAN Transport Layer will cancel (internally) the failed transmission.

CanTp076: For confirmation calls, the function `CanTp_TxConfirmation()` will be provided.

7.2.2 Reception indication

CanTp077: The reception indication function will be called by the CAN Interface module to notify the CAN Transport Layer 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: This reception indication function will be called `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.

CanTp058:
The entire behavior of the CAN Transport Layer will be event triggered, so that CanTp can transfer processes of N-SDU (respectively L-SDU) coming from the PDU Router (respectively CAN Interface) directly.

7.3.1 N-SDU Reception

CanTp079: To optimize communication stack resources, it has been decided to provide the CAN Transport Layer with limited buffering capacity. Therefore, when receiving an SF or an FF N-PDU, the CAN Transport Layer should notify the upper layer (PDU Router) about this reception and request an Rx buffer to process the frame reassembly. These two operations are performed within the `PduR_CanTpProvideRxBuffer()` service.

CanTp166: At the reception of a FF or a SF, the time-out N_Br is started before requesting a Rx buffer. If a buffer has not been provided before the timer elapsed, the communication shall be aborted.

CanTp080: The Rx buffer provided can be smaller than the expected N-SDU data length. In this case, when the current buffer has been filled up with data, the CAN Transport Layer will 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 buffer available to the CAN Transport Layer (i.e. the Rx buffer is locked until CanTp calls either `PduR_CanTpRxIndication()` or `PduR_CanTpProvideRxBuffer()`).

CanTp081: 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`. The CanTp entity will then send a Flow Control N-PDU with overflow status (FC(OVFLW)) and abort the N-SDU reception.

CanTp082: If the upper layer temporarily has no Rx buffer available, the `PduR_CanTpProvideRxBuffer()` function returns `BUFREQ_E_BUSY`. In that case, the CanTp module will suspend the N-SDU reception by sending the next Flow Control N-PDU with status WAIT (i.e. FC(WT)). 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 has to call the service to provide an Rx buffer again during the next processing of the MainFunction.

CanTp083: If the buffer request is delayed again, a new FC(WAIT) must be sent.
At worst, the CanTp module will send a maximum of WFTmax consecutive FC(WAIT) N-PDU. If this number is reached, the CanTp aborts 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.
When the Rx buffer is finally provided the CanTp sends a Flow Control N-PDU with ClearToSend status (FC(CTS)) and carries on the reception of the Consecutive Frame N-PDUs.

CanTp084: When the transport reception session is completed (successfully or not) the CAN Transport Layer will call for a notification service of the upper layer, `PduR_CanTpRxIndication()`.

CanTp064: With regard to FF N-PDU reception, the content of the Flow Control N-PDU depends on the `PduR_CanTpProvideRxBuffer()` service result. Furthermore, it should be noted that when receiving a FF N-PDU, the Flow Control can only be sent after having the result of the `PduR_CanTpProvideRxBuffer()` service.

CanTp067: 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. So 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 will temporarily change the BS value to fill up the buffer reception completely. The desired behavior is described in the following picture

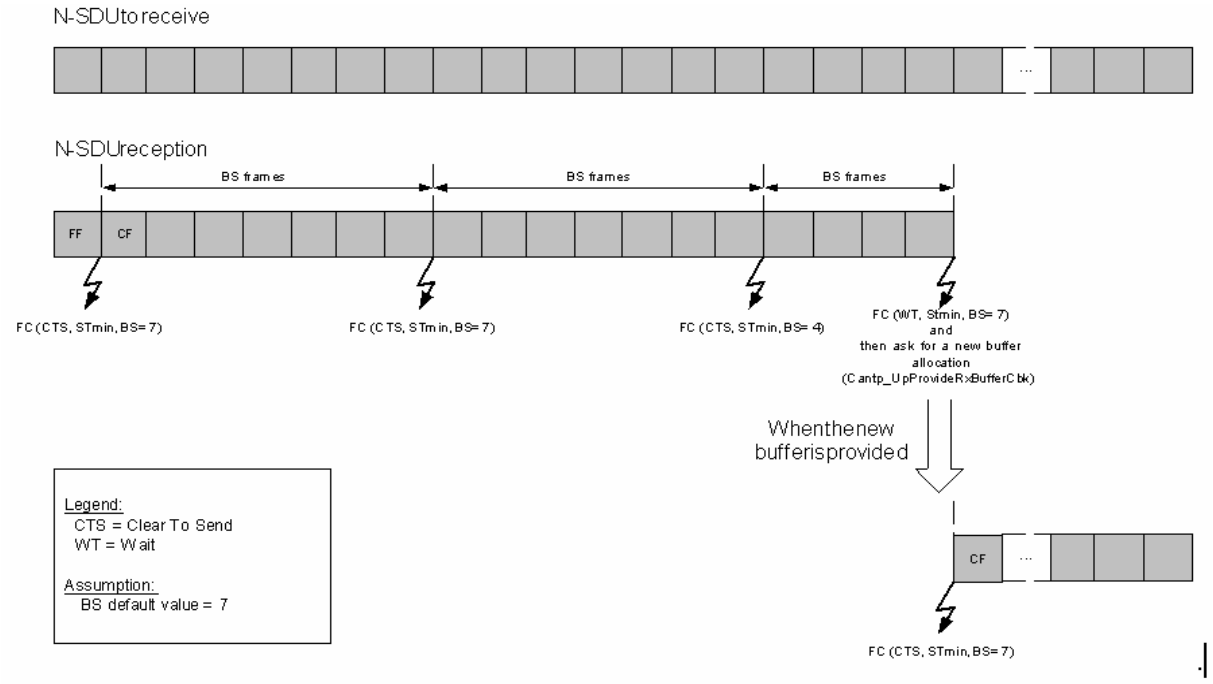


Figure 5: Management of the BS value

7.3.2 N-SDU Transmission

CanTp085: 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.

CanTp110: Only the SduLength information within the returned PduInfoType parameter will be analyzed. The indicator to the payload N-SDU data will be

discarded. This service calls results in at least one call of `PduR_CanTpProvideTxBuffer()`, to request the necessary transmit buffer.

CanTp167: After a transmission request from upper layer, the time-out N_Cs is started before requesting a Tx buffer. If a buffer has not been provided before the timer elapsed, the communication shall be aborted.

CanTp086: The Tx buffer provided can be smaller than the full Tx N-SDU data length. In this case, when the entire content of this buffer has been sent, the CAN Transport Layer will request another buffer by calling the function `PduR_CanTpProvideTxBuffer()`again.

CanTp117: If the data in the Tx buffer cannot be sent completely, the CanTp layer will request a new buffer. Because not all bytes have been sent, the CanTp module will buffer the remaining ones.

CanTp087: If the upper layer cannot provide a Tx 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), the `PduR_CanTpProvideTxBuffer()` function returns `BUFREQ_E_NOT_OK`. The CanTp entity should then 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.

CanTp088: If upper layer temporarily has no Tx buffer available, the `PduR_CanTpProvideTxBuffer()` function returns `BUFREQ_E_BUSY`. In that case, the CanTp module should later retry to receive a buffer. If no buffer 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 CAN Transport Layer will 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 a buffer of free length.

CanTp089: When the Tx buffer is provided, the CAN Transport Layer will resume the transmission of the N-SDU.

CanTp090: When the transport transmission session is successfully completed, the CAN Transport Layer will call a notification service of the upper layer, `PduR_CanTpTxConfirmation()`, with the result NTFRSLT_OK.

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.

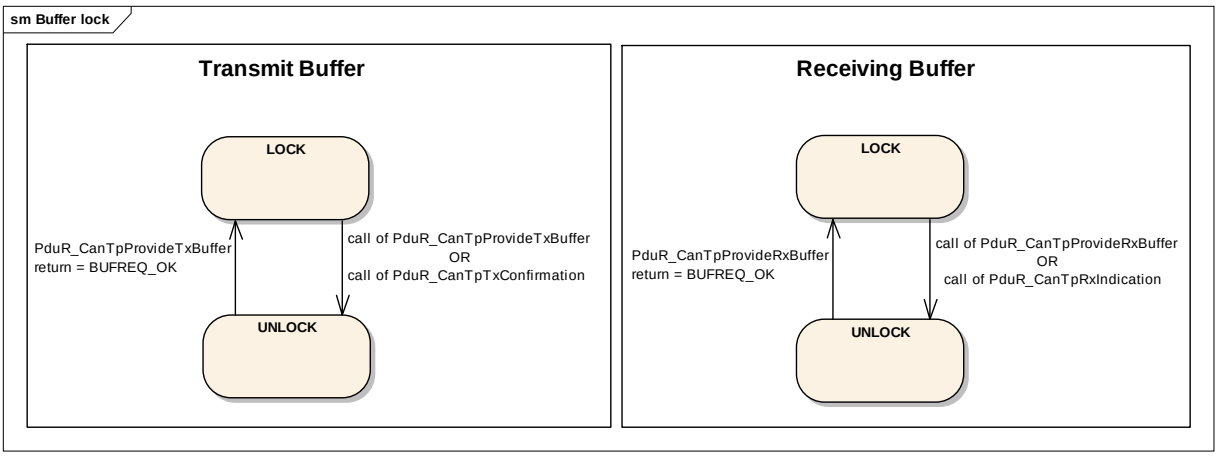
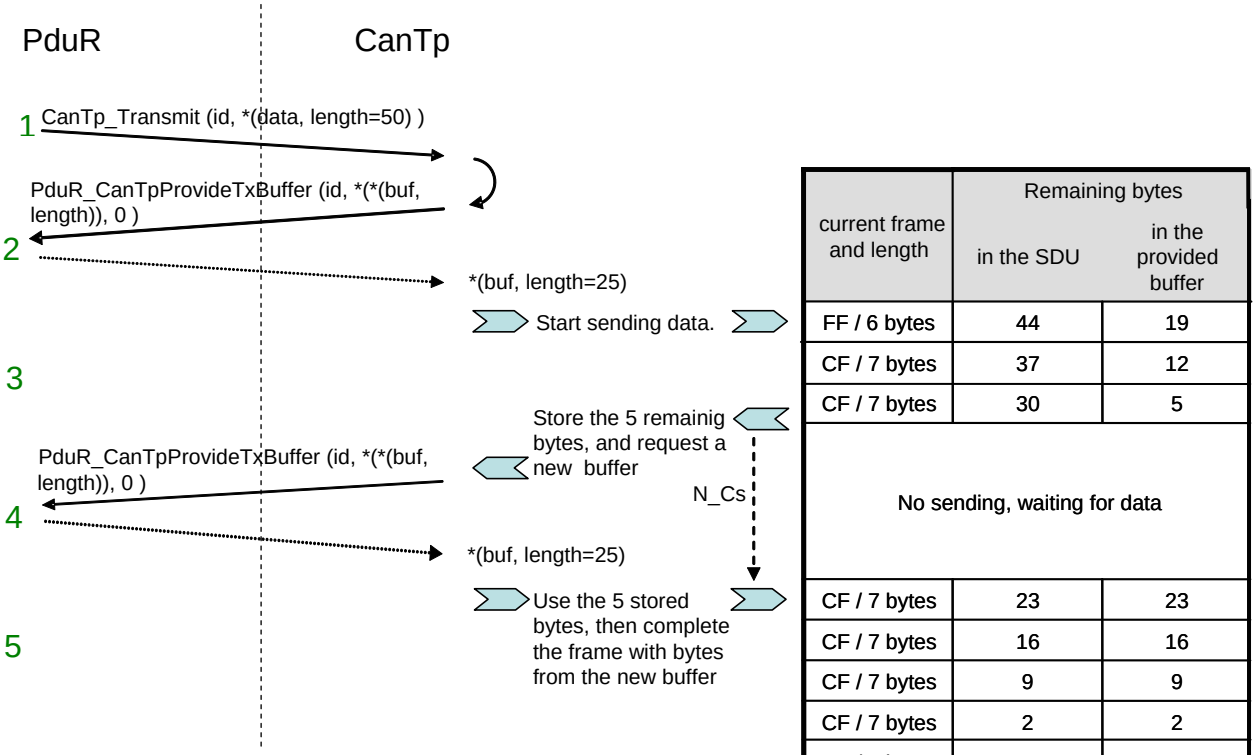


Figure 6: Tx and Rx Buffer locking

Upper layer will lock the buffer when it returns a status `BUFREQ_OK` to a `PduR_CanTpProvideTxBuffer()` or `PduR_CanTpProvideRxBuffer()` call.

The buffer can be unlocked when 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.



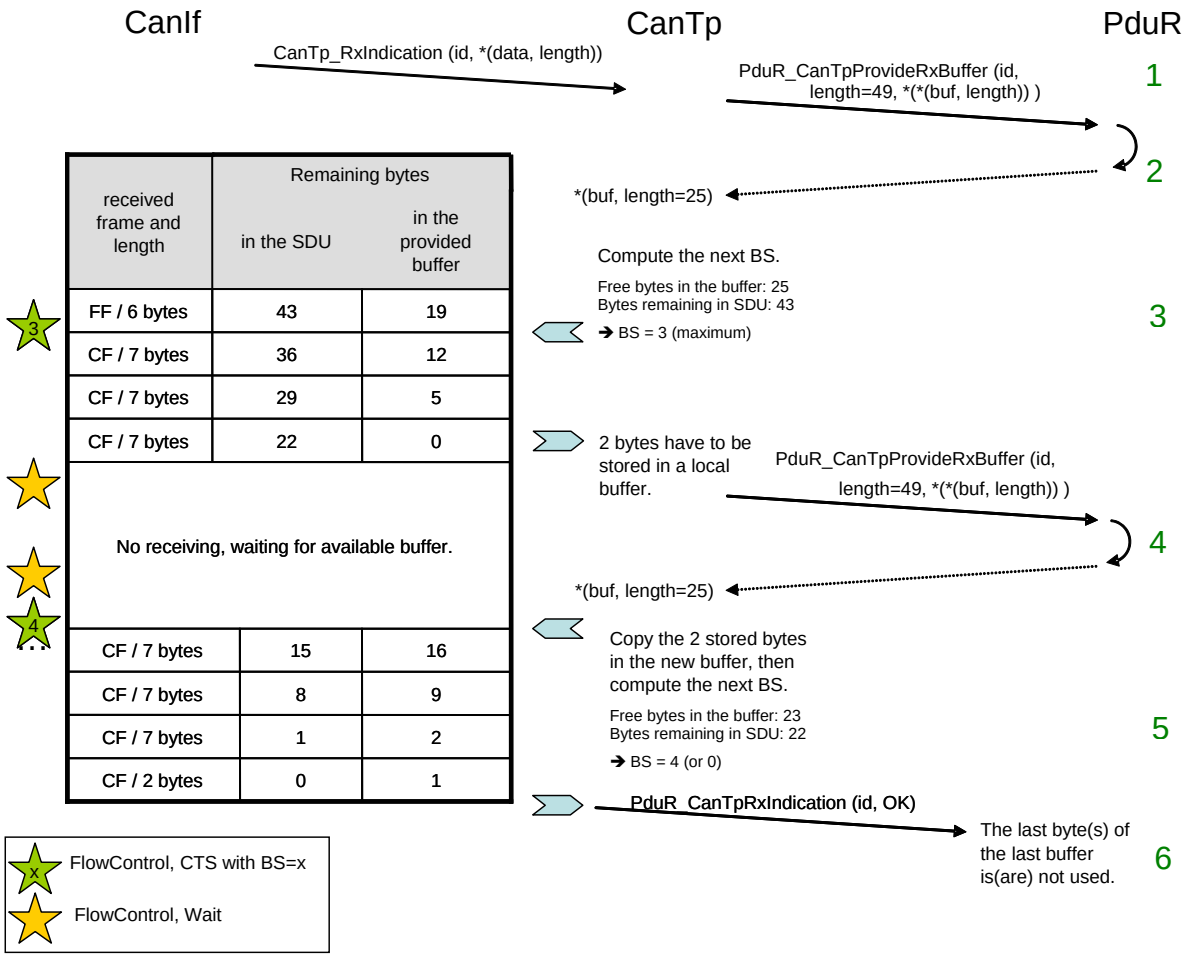
current frame and length	Remaining bytes	
	in the SDU	in the provided buffer
FF / 6 bytes	44	19
CF / 7 bytes	37	12
CF / 7 bytes	30	5
No sending, waiting for data		
CF / 7 bytes	23	23
CF / 7 bytes	16	16
CF / 7 bytes	9	9
CF / 7 bytes	2	2
CF / 2 bytes	0	0

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 by requesting the buffer containing the payload data. The PduR provides a buffer of 25 bytes (by conception in this example, it is not able to provide a buffer of 50 bytes 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 a new buffer 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, by requesting the buffer containing the next payload data. The PduR provides a buffer of 25 bytes
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 a new buffer.
The new buffer 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 buffer (`PduR_CanTpProvideTxBuffer()`) before the current buffer is empty (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 number of BS values (See CanTp067) in the form:
- 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.
It is the responsibility of the upper layer to take care of 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 No Protocol parameter setting services

CanTp091: ISO 15765-2 specification proposes optional primitives for the dynamic setting of some transport protocol internal parameters (STmin and BS) by application. The AUTOSAR CAN Transport Layer does not support these services. Thus, STmin and BS values should only be set statically N-SDU by N-SDU.

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.

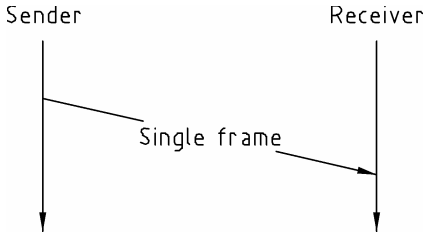


Figure 9: Example of single part message

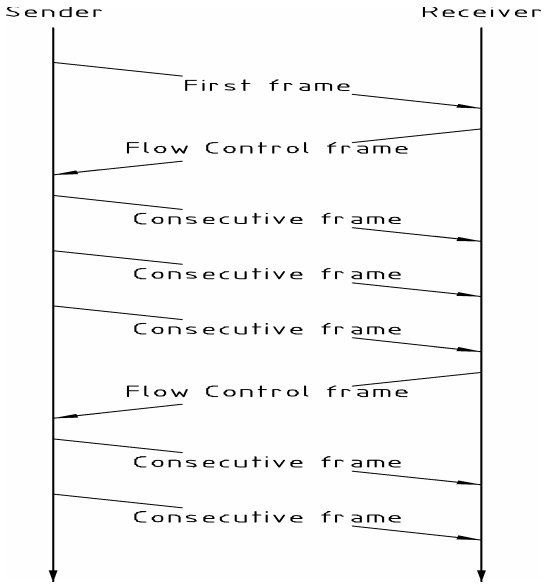


Figure 10: Example of multiple parts message

CanTp092: 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 [12]). Nevertheless, the CanTp module provides 1 to n communication (i.e. functional addressing [12]), in the form of functionality to SF N-PDUs (and only SF N-SDU). Thus, the configuration must check whether it is only SF N-PDUs that have been configured with a functional addressing property.

CanTp093: In spite of these checks if a multiple segmented session occurs (on both receiver and sender side) with a handle whose communication type is functional, the CanTp module will reject the request and generate a development error.

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 should only be linked to one CAN LSduId that is used to transmit SF, FF, FC and CF frames.

CanTp112: 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.

CanTp094: 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. Thus the CAN LSduId of a FC frame combined with its N_TA value (e.g. the N_AI) may only identify one CAN NSduId.

CanTp095: 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. Therefore, in extended addressing N-PDU reception, the CanTp module will extract the N-TA value to establish the related N-SDU.

The following figure summarizes these discussions.

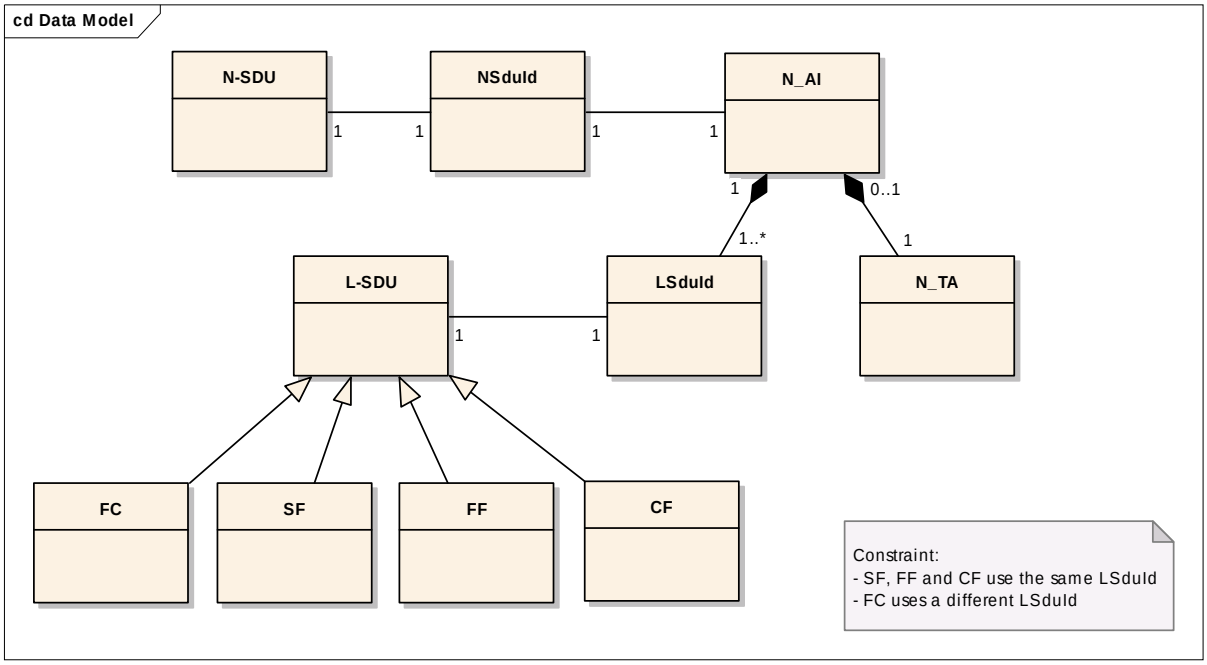


Figure 11: Possible links between NSduId and LSduId

7.3.7 Concurrent connection

CanTp096: 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).

CanTp120: Concurrent connections can be configured to the CAN Transport Layer. 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.

The CAN Transport Layer should 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 CAN Transport Layer will 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: If the configured receiving connection channel is in use (state `CANTP_RX_PROCESSING`), on reception of new data (First Frame reception) the CAN Transport Layer will abort the reception in progress and process the received frame as the start of a new reception.

7.3.8 N-PDU padding

CanTp114: 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 will be configurable at pre-compile time, by using parameter `CANTP_PADDING_ACTIVATION`.

CanTp040: If the `CANTP_PADDING_ACTIVATION` parameter is set to `ON`, the CAN Transport Layer will only transfer N-PDU with a length of eight bytes (i.e. DLC = 8) between the CanTp and the CanIf. Thus, a received N-PDU shorter than 8 bytes will be considered corrupt.

CanTp098: If the `CANTP_PADDING_ACTIVATION` parameter is set to `CANTP_OFF`, the CAN Transport Layer module shall check the frame data length. If a frame is received with an unexpected datalength (check only for too short DLCs), the frame shall be ignored.

CanTp116: In both padding and no padding modes, only used data bytes should be transfered to the upper layer.

CanTp059: To improve transfer time, unused byte(s) will be sent without any initialization.

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: The table below defines the CAN Transport Layer behavior if unexpected frames are received, in consideration of the actual CanTp internal status (CanTp status) and the requirement to not support full-duplex communication. 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

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.

CanTp012: 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.

CanTp008: Errors and exceptions (either development or production errors) must not modify the current CAN Transport Layer module macro state (see Figure 4: CAN Transport Layer life cycle). The CanTp should simply report the error event and then, in case of production error, the Diagnostic Event Manager module (via the Function Inhibition Manager) will perform the appropriate action (e.g. status modification of the calling module).

Values for production code Event Ids are assigned externally by the configuration of the Dem. They are published in the file Dem_IntErrId.h and included via Dem.h.

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	Development	CANTP_E_INVALID_RX_ID	0x40

identifier)			
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 PDU (e.g. a transmit N-SDU has a length equal to zero)	Development	CANTP_E_INVALID_TX_LENGTH	0x70
Invalid data length of the receive PDU (e.g. a receive FF N-PDU has a FF_DL equal to zero)	Development	CANTP_E_INVALID_RX_LENGTH	0x80
CanTp_Transmit() is called for a configured Tx I-Pdu with functional addressing and the length parameter indicates, that the message can not be sent with a SF	Development	CANTP_E_INVALID_TATYPE	0x90
Requested operation is not supported	Production	CANTP_E_OPER_NOT_SUPPORTED	Assigned by DEM
Another error occurred during a reception or a transmission	Production	CANTP_E_COMM	Assigned by DEM

7.5 Error detection

CanTp006: The detection of development errors is configurable (*ON* / *OFF*) at pre-compile time.
The switch *CANTP_DEV_ERROR_DETECT* (see chapter 10) should activate or deactivate the detection of all development errors.

CanTp132: If the *CANTP_DEV_ERROR_DETECT* 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.

CanTp133: The detection of production code errors cannot be switched off.

CanTp161: A static status variable, denoting whether a BSW module is initialized, should be initialized with value 0 before any APIs of the BSW module are called.
The initialization function of the BSW modules will set the static status variable to a value not equal to 0.
This variable is used to check if the module has been initialized before calling an API.

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 *CANTP_DEV_ERROR_DETECT* is set (see chapter 10).

CanTp013: 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: To report development errors the CanTp will use the Development Error Tracer service [8]:
`void Det_ReportError(ModuleId, InstanceId, ApiId, ErrorId)`
Where:

- `ModuleId` is the basic software module identifier. It is a uint16 constant
- `InstanceId` is the identifier (uint8) of the index based instance of a module, starting from 0, If the module is a single instance module it shall pass 0 as the InstanceId
- `ApiId` is a uint8 constant specified within this specification document (This is the service Id of the calling primitive).
- ErrorId is a uint8 constant that represents the error identifier. It is specified in chapter 7.4 of this document. However, its value can be #defined externally in the module's header file.

CanTp115: Thus, the header file of the CAN Transport Layer, `CanTp.h`, will 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.

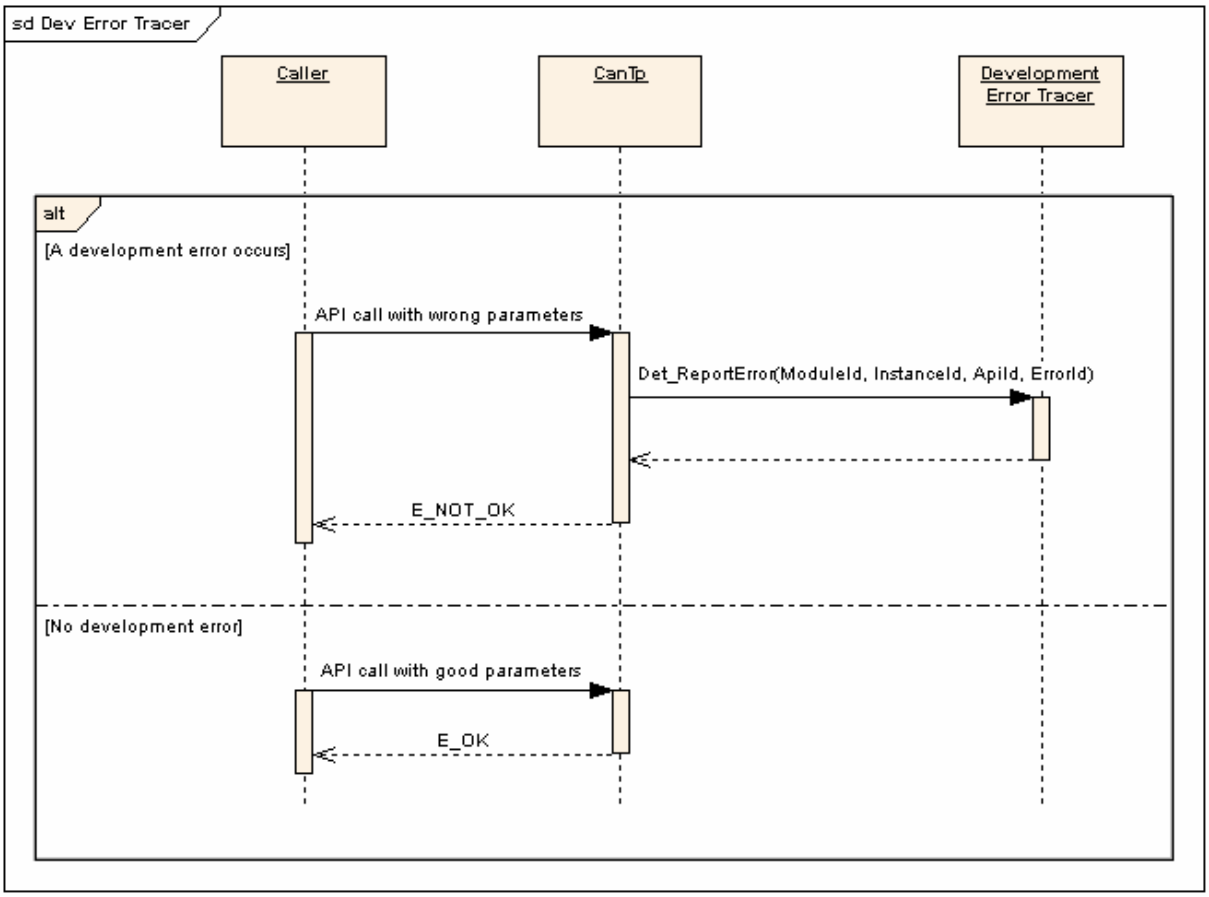


Figure 12: Development error reporting

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.

CanTp135: This module is a standard AUTOSAR module which is available in production code and whose functionality is specified in the AUTOSAR project [2].

CanTp100: To report production errors the CanTp will use the API services of the DEM module software specification [11]. Therefore, this DEM module dumps the error into the “error memory” and informs the FIM module, which has a pre-defined reaction for the ECU (e.g. disabling ECU functions, switching ECU mode, etc).
After a task is completed successfully by calling PduR_TxConfirmation() or PduR_RxIndication(), the DEM should be called with EventStatus=DEM_EVENT_STATUS_PREPASSED. If the task was aborted (e.g. As, Bs, Cs, Ar, Br, Cr timeout) the DEM should be called with EventStatus=DEM_EVENT_STATUS_PREFAILED.
A debounce algorithm of type *Counter based* has to be used in DEM for CanTp errors.

8 API specification

8.1 Imported types

8.1.1 Standard types

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

- Std_Types.h
- ComStack_Types.h

This types are :

- Std_ReturnType
- PduIdType
- PduInfoType
- NotifResultType
- BufReq_ReturnType

CanTp118: 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.

CanTp016: Thus, AUTOSAR standard types will be used without any type redefinition.

8.1.2 CanTp types

No specific types are defined in the CAN Transport Layer.

CanTp002: If, for implementation reasons, some types have to be defined. These types will be labeled as follows: CanTp_<TypeName>Type, where <TypeName> is name of this type and adheres to the rules:

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

Implementation specific types should not be "visible" outside of CanTp. Otherwise the complete architecture would be corrupted.

8.2 Type definitions

No specific types are defined in the CAN Transport Layer.

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

Service name:	CanTp_Init
Syntax:	`void CanTp_Init ( void )`
Service ID [hex]:	0x01
Sync/Async:	Synchronous
Reentrancy:	Non reentrant
Parameters (in):	`none` --
Parameters (out):	`none` --
Return value:	`none` --
Description:	**CanTp018:** After power up, CanTp is in a state called CANTP_OFF. In this state, the CanTp is not yet configured and therefore cannot perform any communication task. This service uses the given configuration set, to initialize all global variables of the CAN Transport Layer and set it in the idle state (state = CANTP_ON, but neither transmission nor reception are in progress). It 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.
Caveats:	The calling of this service is mandatory before using the CanTp module for further processing.
Configuration:	--

8.3.2 CanTp_ GetVersionInfo

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 (out):	`versioninfo` Indicator as to where to store the version information of this module.
Return value:	none --
Description:	**CanTp162:** This service returns the version information of this module. The version information includes: - Module Id - Vendor Id

	- Vendor specific version numbers (BSW00407). **CanTp163:** This function should be configurable in the pre compile time (`On/Off`) by the configuration parameter: `CANTP_VERSION_INFO_API` Hint: If source code for caller and callee of this function is available this function should be realized as a macro. The macro should be defined in the modules' header file. This interface can be called before initialization of CanTp module.
Caveats:	--
Configuration:	`CANTP_VERSION_INFO_API`

8.3.3 CanTp_Shutdown

Service name:	CanTp_Shutdown
Syntax:	`void CanTp_Shutdown( void )`
Service ID [hex]:	0x02
Sync/Async:	Synchronous
Reentrancy:	Non reentrant
Parameters (in):	none --
Parameters (out):	none --
Return value:	none --
Description:	**CanTp011:** This service closes all pending transport protocol connections, frees all resources and sets the corresponding CanTp module into the CANTP_OFF state. However, it will not raise notification about the pending frame transmission or reception. If an internal error occurred while stopping the CanTp the DEM module will be notified.
Caveats:	None
Configuration:	--

8.3.4 CanTp_Transmit

Service name:	CanTp_Transmit	
Syntax:	`Std_ReturnType CanTp_Transmit ( PduIdType CanTpTxSduId, const PduInfoType* CanTpTxInfoPtr )`	
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
	`CanTpTxInfoPtr`	An indicator of a structure with CAN N-SDU related data: indicator of a CAN N-SDU buffer and the length of this buffer.
Parameters (out):	`None`	--
Return value:	`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:	**CanTp103:** This service is used to request the transfer of segmented data. If data length is less than 7 or 6 (depending on normal or extended addressing format), a SF N-PDU is sent. However, if data length is greater than 7 or 6, a multiple frame transmission session is initiated. When the transmit request has been completed, the CanTp notifies the upper layer by calling the `PduR_CanTpTxConfirmation` callback. If an error occurred (over flow, N_As timeout, N_Bs timeout and so on), the transmit request is aborted and the `PduR_CanTpTxConfirmation` callback is called with the appropriate error result value. If the `CanTp_Transmit` service is called for a N-SDU identifier which is being used in a currently running CAN Transport Layer session, the CAN Transport Layer should reject the request. 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 of the `CanTpTxInfoPtr` structure directly. 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.	
Caveats:	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.	
Configuration:	--	

8.3.5 Main Function

Service name:	CanTp_MainFunction
Syntax:	`void CanTp_MainFunction(void)`
Service ID:	0x06

Sync/Async:	Synchronous
Reentrancy:	non reentrant
Parameters (in):	None --
Parameters (out):	None --
Return value:	none --
Description:	**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 FRT module according of the call period needed.
Caveats:	--
Configuration:	CANTP_MAIN_FUNCTION_PERIOD

8.4 Call-back notifications

The following is a list of functions provided for lower layer modules. The function prototypes of the callback functions will be provided in the file `CanTp_Cbk.h`

8.4.1 CanTp_RxIndication

Service name:	CanTp_RxIndication	

Syntax:	`void CanTp_RxIndication`
	`(`
	`  PduIdType  CanTpRxPduId,`
	`  const PduInfoType*  CanTpRxPduPtr`
	`)`	

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
	`CanTpRxPduPtr`	Indicator of structure with received L-SDU (payload) and data length
Parameters (out):	`none`	--
Return value:	`none`	--
Description:	**CanTp019:** This function is called by the CAN Interface after a successful reception of a Rx CAN L-PDU. This callback service is called by the Can Interface and implemented by the CanTp. It will be called in the case of a Rx indication of the CAN driver. 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.	
Caveats:	This function could be called if an interrupt occurs (from the CAN receive interrupt). This function will be implemented using a Pre-compile macro	
Configuration:	--	

8.4.2 CanTp_TxConfirmation

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 (out):	`none` --
Return value:	`none` --
Description:	**CanTp020:** This function is called by the CAN Interface after the TP related CAN Frame (SF, FF, CF, FC) has been transmitted through the CAN network. All transmitted CAN frames belonging to the CAN Transport Layer will be confirmed by this function.
Caveats:	This function could be called if an interrupt occurs (from the CAN transmit interrupt). This function will be implemented using a Pre-compile macro
Configuration:	--

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.

API function	*Module*	*Description*
PduR_CanTpProvideTxBuffer	PduR	Request a buffer to upper layers where the CanTp will read the data to be transmitted.
PduR_CanTpProvideRxBuffer	PduR	Request a buffer to upper layers where the CanTp will write the received data.
PduR_CanTpTxConfirmation	PduR	Confirmation status of a N-SDU transmitted to upper layers.
PduR_CanTpRxIndication	PduR	Indication status of a received N-SDU to upper layers.
CanIf_Transmit	CanIf	Request the transmission of an N-PDU to the CAN Interface Layer.
Dem_ReportErrorStatus	Dem	Report production error to the DEM

8.5.2 Optional Interfaces

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

API function	*Module*	*Description*	*Configuration parameter (description see chapter 10)*
Det_ReportError	Det	Development error notification	CANTP_DEV_ERROR_DETECT

9 Sequence diagrams

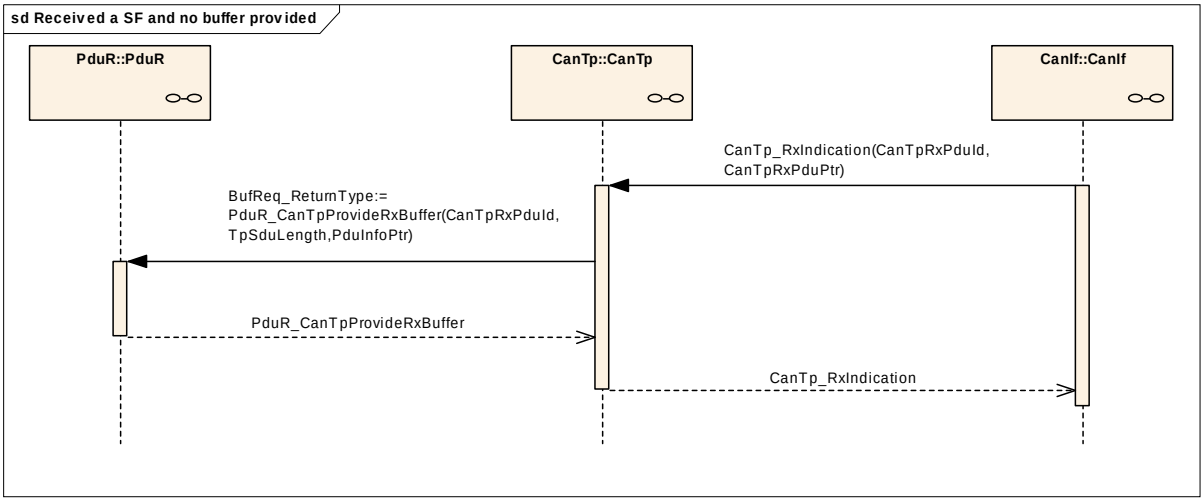
The goal of this chapter is to make it easier to understand the CAN Transport Layer by describing most of the more frequent and complicated use cases. Thus, the following diagram sequences are not exhaustive and do not reflect all the specified API possibilities.

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

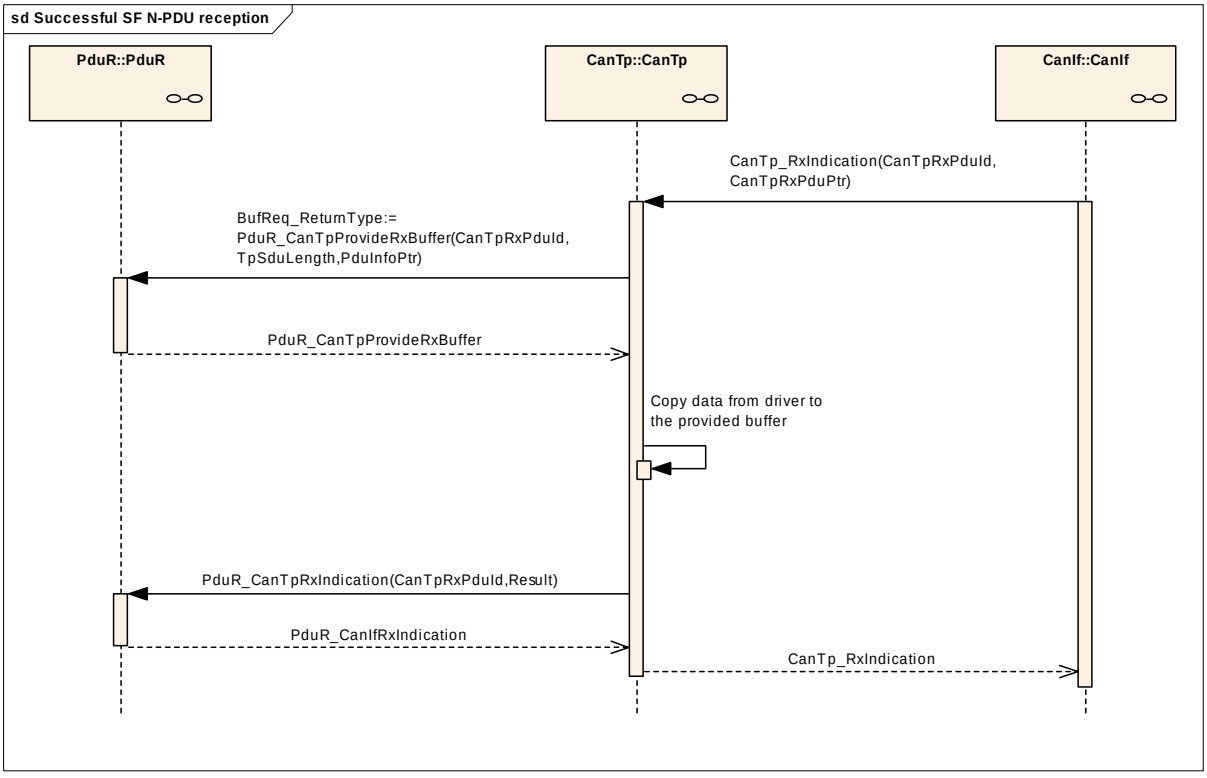
Transition	Name	Description
2	`PduR_CanTpProvideRxBuffer( CanTpRxSduId, TpSduLength, PduInfoPtr )`	The CAN Transport Layer performs an ID translation and extracts the useful data length from the N-PDU payload. It then asks its upper layer to provide a buffer for this incoming data with a PduR_CanTpProvideRxBuffer 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_RxIndication` function without copying any data.

9.2 Successful SF N-PDU reception

9.2.1 Assumptions

- All input parameters are OK
- The N-SDU data length is smaller than or equal to 7 bytes (6 bytes in the case of extended addressing format)
- The SF N-PDU is successfully received

9.2.2 Sequence diagram



Note: This sequence diagram demonstrates the working of the CAN_Tp module only. However, if the whole system is considered during such reception, more modules are

involved. Since this reception can be triggered in the context of CAN ISR, the CAN_Tp operation should be as short as possible.

9.2.3 Transition description

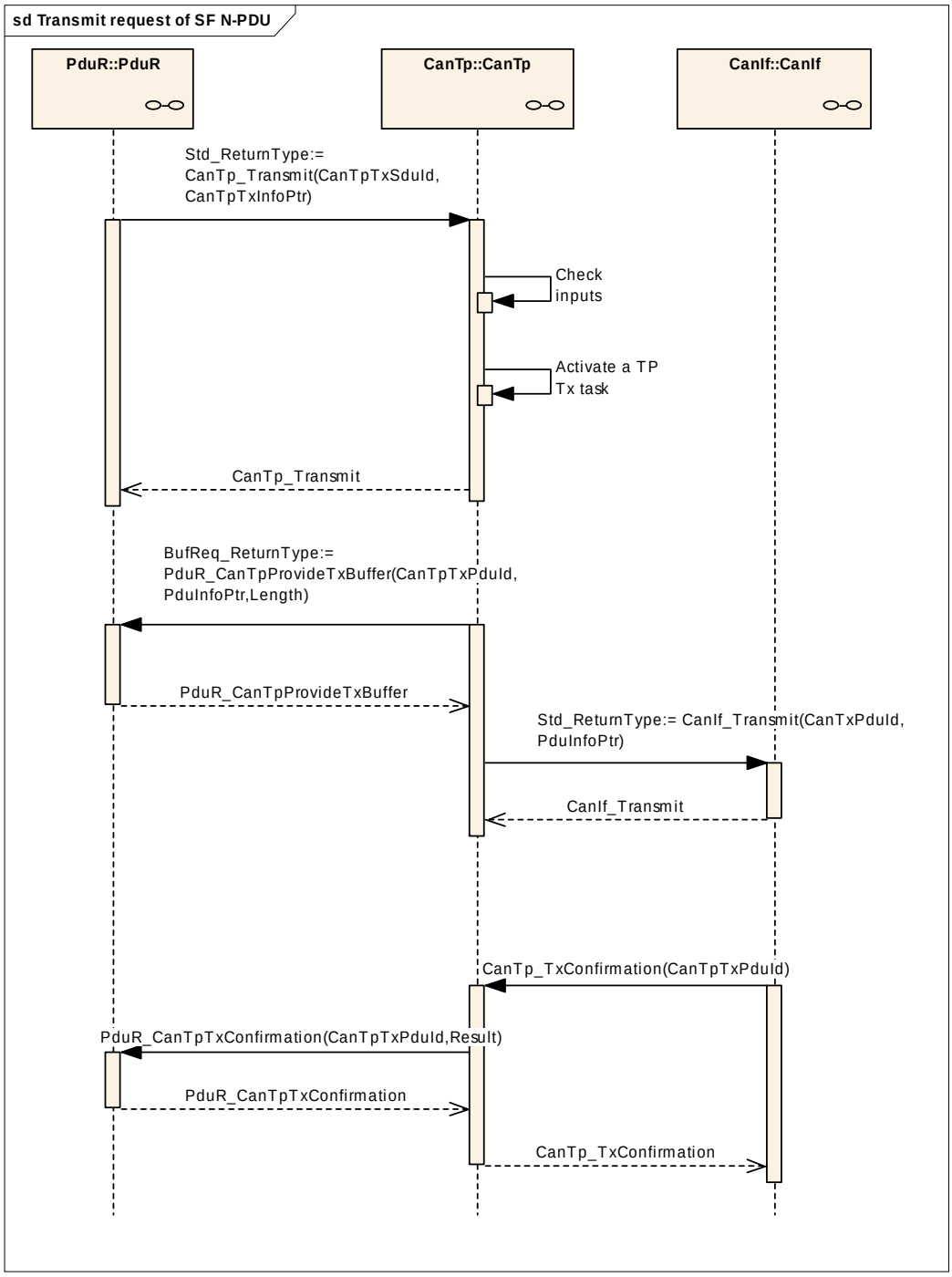
Transition	*Name*	*Description*
1	`CanTp_RxIndication ( CanTpRxPduId, CanTpRxPduPtr )`	When the lower layer receives a frame (here a single frame), it notifies CanTp by means of a CanTp_RxIndication callback. CanTpRxPduId represents the ID of the L-PDU that has been received, and CanTpRxPduPtr indicates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

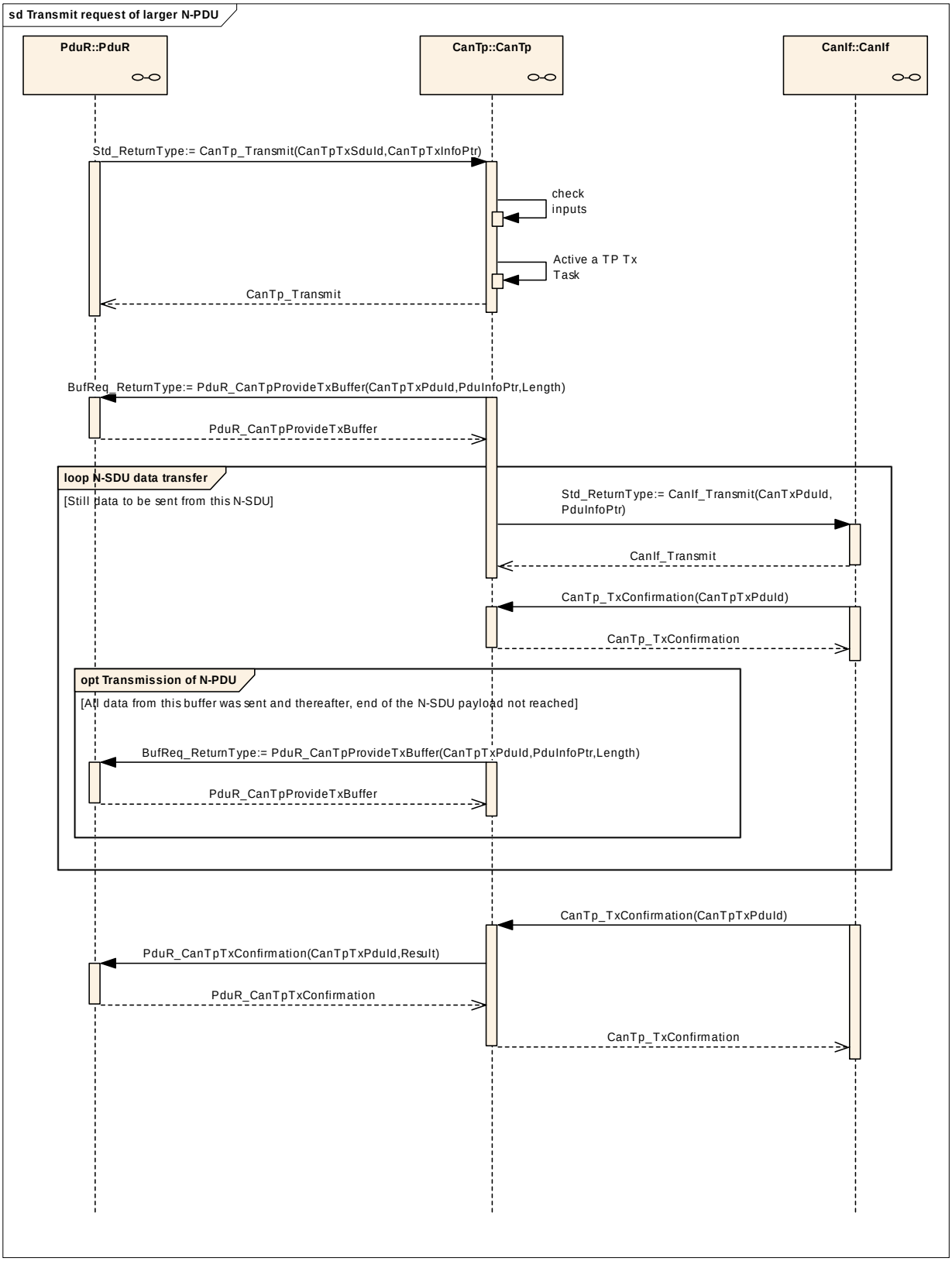
Transition	Name	Description
1	`CanTp_Transmit( CanTpTxSduId, CanTpTxInfoPtr )`	The PDU Router needs (because of a request from the diagnostic controller module or a 1:1 TP routing – i.e. the PduR itself) to transmit an I-PDU that requires transport protocol functionality and whose data can be refered 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_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).
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_TxConfirmatio n( CanTpTxPduId, )`	The N-PDU is successfully transmitted.
8	`PduR_CanTpTxConfirm ation ( 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



9.4.3 Transition description

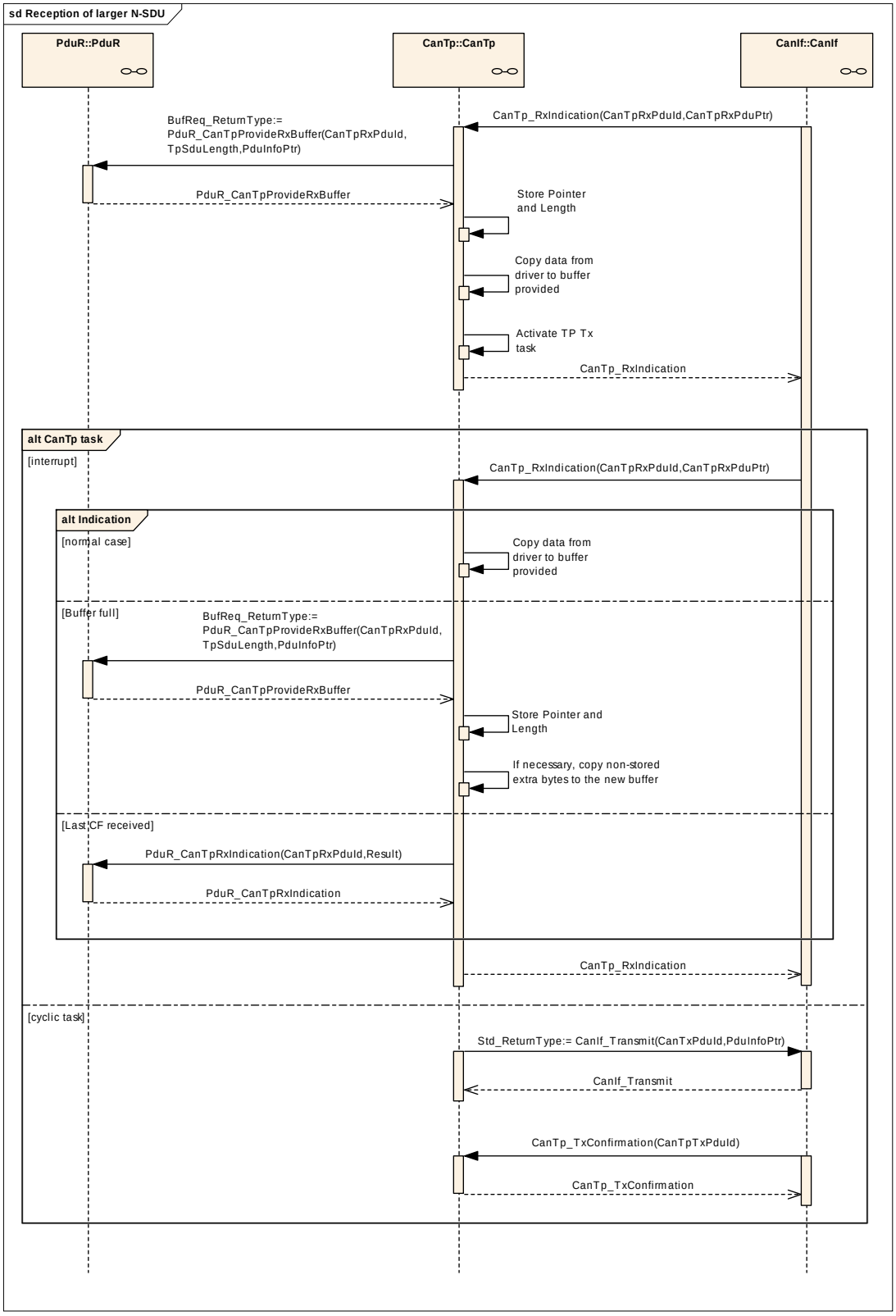
Transition	Name	Description
1	`CanTp_Transmit ( CanTpTxSduId, CanTpTxInfoPtr )`	The PDU Router needs (because of a request from the diagnostic controller module or a 1:1 TP routing – i.e. the PduR itself) to transmit an I-PDU that requires transport protocol functionality and whose data can be referred 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_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).
3	`BUFREQ_OK`	The upper layer allocates and locks the required Tx buffer. Then returns `BUFREQ_OK`.
4	`CanIf_Transmit ( CanTxPduId, PduInfoPtr )`	Within the task, CanTp calls the CAN Interface by using `CanIf_Transmit`, where `CanTxPduId` identifies the L-SDU (a translation has to be preformed between the N-SDU Id used by CanTp and the L-SDU Id used by CAN Interface), and `PduInfoPtr` indicator data and their length.
5	`CanTp_TxConfirmation( CanTpTxPduId, )`	CanTp awaits a confirmation from the CAN Interface (`CanTp_TxConfirmation`)
6	`PduR_CanTpProvideTxBuffer ( 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_CanTpTxConfirmation ( 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



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[3]	Refer here to (a) variant(s).
	Link time	see[4]	Refer here to (a) variant(s).
	Post Build	see[5]	Refer here to (a) variant(s).
Scope	Describes the scope of the parameter. The scope describes the impact of the configuration parameter: Does the setting affect only one instance of the module (instance), all instances of this module (module), the ECU or a network? Possible values of scope : instance, module, ECU, network		
Dependency	Describes the dependencies with respect to the scope.		

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

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

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

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

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

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

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

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

10.2 Containers and configuration parameters

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

10.2.1 Variants

Variant 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 CanTpConfiguration

SWS Item	
Container Name	CanTpConfiguration
Description	This container contains the configuration (parameters) of the CAN Transport Layer module.
Configuration Parameters	

Name	CANTP_DEV_ERROR_DETECT		
Description	Switches the Development Error Detection and Notification ON or OFF		
Type	#define		
Unit	--		
Range	STD_ON	enable	
	STD_OFF	disable	
Configuration Class	***Pre-compile***	x	All Variants
	Link time	--	--
	Post Build	--	--
Scope	Module		
Dependency	--		

Name	CANTP_MAIN_FUNCTION_PERIOD		
Description	Allow to configure the time for the MainFunction (in ms). Please note: This configuration value shall be equal to the value in the ScheduleManager module.		
Type	#define Variant 1 uint8 Variant 2		
Unit	ms		
Range	0…255		
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	--	--
	Post Build	L	Variant 2
Scope	ECU		
Dependency	--		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
none	--	--

10.2.3 RxNsdu

SWS Item	**CanTp137:**
Container Name	RxNsdu
Description	The following parameters should be configured for each CAN N-SDU the CanTp module has to receive.
Configuration Parameters	

Name	CANTP_RXNSDU_ID		
Description	Unique identifier of a structure that contains all useful information for the process: reception of an RxNsdu.		
Type	#define		
Unit	--		
Range	0..*	Max: number of Rx N-Sdu.	
Configuration Class	***Pre-compile***	x	All Variants
	Link time	--	--
	Post Build	--	--
Scope	module		
Dependency	--		

Name	CANTP_PADDING_ACTIVATION		
Description	Defines if the receive frame uses padding or not.		
Type	#define		
Unit	--		
Range	CANTP_ON	The N-PDU received uses padding for SF, FC and the last CF. (N-PDU length is always 8 bytes)	
	CANTP_OFF	The N-PDU received does not use padding for SF, CF and the last CF. (N-PDU length is dynamic)	
Configuration Class	***Pre-compile***	x	All Variants
	Link time	--	--
	Post Build	--	--
Scope	module		
Dependency	--		

Name	CANTP_RXNPDU_ID
Description	The N-PDU identifier attached to the RxNsdu is identified by CANTP_RXNSDU_ID. 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_SA value (first data byte of SF or FF frames).
Type	#define
Unit	--

Range	`0..*`	Max: number of Rx N-Pdu.	
Configuration Class	***Pre-compile***	x	All Variants
	Link time	--	--
	Post Build	--	--
Scope	module		
Dependency	--		

Name	`CANTP_TXFC_NPDU_ID`		
Description	The N-PDU identifier attached to the FC N-PDU of this RxNsdu is identified by `CANTP_RXNSDU_ID`. Each RxNsdu identifier is linked to only one Tx FC N-PDU identifier. 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 the N_TA value (first data byte of FC frames).		
Type	`#define`		
Unit	--		
Range	`0..*`	Max: number of Tx N-Pdu.	
Configuration Class	***Pre-compile***	x	All Variants
	Link time	--	--
	Post Build	--	--
Scope	module		
Dependency	--		

Name	`CANTP_TA_TYPE`		
Description	Declares the communication type for this RxNsdu.		
Type	`#define`		
Unit	`--`		
Range	`CANTP_PHYSICAL`	used for 1 to 1 communication	
	`CANTP_FUNCTIONAL`	used for 1..n communication (is only supported for SF N-PDU).	
Configuration Class	***Pre-compile***	x	All Variants
	Link time	--	--
	Post Build	--	--
Scope	module		
Dependency	--		

Name	`CANTP_ADDRESSING_FORMAT`		
Description	Declares which communication addressing format is supported for this RxNsdu.		
Type	`#define`		
Unit	`--`		
Range	`CANTP_STANDARD`	to use normal addressing format.	
	`CANTP_EXTENDED`	to use extended addressing format (the N_TA container of this RxNsdu will be used).	
Configuration Class	***Pre-compile***	x	All Variants
	Link time	--	--
	Post Build	--	--
Scope	module		
Dependency	--		

Name	`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 subsequent N-PDUs. For further details of this parameter value see ISO 15765-2		

	specification [10]. Note: for reasons of buffer length, the CAN Transport Layer can adapt the BS value within the limit of this maximum BS.		
Type	`#define Variant 1` `uint8 Variant 2`		
Unit	`N-PDU`		
Range	--	BS value	
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	--	--
	Post Build	L	Variant 2
Scope	module		
Dependency	`--`		

Name	`CANTP_STMIN`		
Description	Sets the duration of the minimum time the CanTp sender should wait between the transmissions of two CF N-PDUs. For further details of this parameter value see ISO 15765-2 specification [12].		
Type or Unit	`#define Variant 1` `uint8 Variant 2`		
Range	`--`	STmin value	
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	--	--
	Post Build	L	Variant 2
Scope	module		
Dependency	`--`		

Name	`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. `CANTP_WFTMAX` 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. For more description see [12] (chapter “Maximum number of FC.Wait frame transmissions”).		
Type	`#define Variant 1` `uint8 Variant 2`		
Unit	Number of Flow Control wait N-PDUs		
Range	--	WFTmax value	
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	--	--
	Post Build	L	Variant 2
Scope	module		
Dependency	--		

Name	`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.		
Type	`#define`		
Unit	`bytes`		
Range	--	DL value	
Configuration Class	***Pre-compile***	x	All Variants
	Link time	--	--
	Post Build	--	--

Scope	module
Dependency	--

Name	CANTP_NAR		
Description	Value in ms of the N_Ar timeout. N_Ar is the time for transmission of a CAN frame (any N_PDU) on the receiver side.		
Type	#define Variant 1 Uint16 Variant 2		
Unit	ms		
Range	--	N_Ar value	
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	--	--
	Post Build	L	Variant 2
Scope	module		
Dependency	--		

Name	CANTP_NBR		
Description	Value in ms 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.		
Type	#define Variant 1 Uint16 Variant 2		
Unit	ms		
Range	--	N_Br value	
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	--	--
	Post Build	L	Variant 2
Scope	module		
Dependency	--		

Name	CANTP_NCR		
Description	Value in ms of the N_Cr timeout. N_Cr is the time until reception of the next Consecutive Frame N_PDU.		
Type	#define Variant 1 Uint16 Variant 2		
Unit	ms		
Range	--	N_Cr value	
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	--	--
	Post Build	L	Variant 2
Scope	module		
Dependency	--		

Name	CANTP_RX_CHANNEL		
Description	Link to the Rx connection channel, which has to be used for receiving this N-PDU.		
Type	#define		
Unit	--		
Range	0..*	Max: number of Rx channel	
Configuration Class	***Pre-compile***	x	All Variants
	Link time	--	--
	Post Build	--	--
Scope	module		
Dependency	--		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
N_Sa	0..1	module
N_Ta	0..1	module

10.2.4 TxNsdu

SWS Item	**CanTp138:**
Container Name	TxNsdu
Description	The following parameters must be configured for each CAN N-SDU, the CanTp module has to transmit.
Configuration Parameters	

Name	CANTP_TXNSDU_ID		
Description	Unique identifier to a structure that contains all useful information to process the transmission of a TxNsdu.		
Type	#define		
Unit	--		
Range	0..*	Max: number of Tx N-Sdu	
Configuration Class	***Pre-compile***	x	All Variants
	Link time	--	--
	Post Build	--	--
Scope	module		
Dependency	--		

Name	CANTP_PADDING_ACTIVATION	
Description	Defines if the transmit frame use padding or not.	
Type	#define	
Unit	--	
Range	CANTP_ON	The transmit N-PDU uses padding for SF, FC and the last CF. (N-PDU length is always 8 bytes)
	CANTP_OFF	The transmit N-PDU does not use padding for SF, CF and the last CF. (N-PDU length is dynamic)
Configuration Class	***Pre-compile***	x / All Variants
	Link time	-- / --
	Post Build	-- / --
Scope	module	
Dependency	--	

Name	CANTP_TXNPDU_ID
Description	N-PDU identifier attached to the TxNdsu identified by CANTP_TXNSDU_ID. Each TxNsdu identifier is linked to one SF/FF/CF N-PDU identifier only. However, 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 means of the N_TA value (first data byte of SF or FF frames).
Type	#define
Unit	--

Range	`0..*`	Max: number of Tx N-Pdu.	
Configuration Class	***Pre-compile***	x	All Variants
	Link time	--	--
	Post Build	--	--
Scope	module		
Dependency	--		

Name	`CANTP_RXFC_NPDU_ID`		
Description	N-PDU identifier attached to the FC N-PDU of this TxNsdu identified by CANTP_TXNSDU_ID. 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_SA value (first data byte of FC frames).		
Type	`#define`		
Unit	--		
Range	0..*	Max: number of Rx N-Pdu.	
Configuration Class	***Pre-compile***	x	All Variants
	Link time	--	--
	Post Build	--	--
Scope	module		
Dependency	--		

Name	`CANTP_TA_TYPE`		
Description	Declares the communication type of this TxNsdu.		
Type	`#define`		
Unit	`--`		
Range	`CANTP_PHYSICAL`	used for 1 to 1 communication	
	`CANTP_FUNCTIONAL`	used for 1..n communication (is only supported for SF N-PDU).	
Configuration Class	***Pre-compile***	x	All Variants
	Link time	--	--
	Post Build	--	--
Scope	module		
Dependency	--		

Name	`CANTP_ADDRESSING_MODE`		
Description	Declares which communication addressing format is supported for this TxNsdu.		
Type	`#define`		
Unit	`--`		
Range	`CANTP_STANDARD`	to use normal addressing format.	
	`CANTP_EXTENDED`	to use extended addressing format (the N_TA container of this TxNsdu will be used).	
Configuration Class	***Pre-compile***	x	All Variants
	Link time	--	--
	Post Build	--	--
Scope	module		
Dependency	--		

Name	`CANTP_NAS`		
Description	Value in ms 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.		
Type	`#define Variant 1`		

	Uint16 Variant 2		
Unit	ms		
Range	--	N_As value	
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	--	--
	Post Build	L	Variant 2
Scope	module		
Dependency	--		

Name	CANTP_NBS		
Description	Value in ms of the N_Bs timeout. N_Bs is the time of transmission until reception of the next Flow Control N_PDU.		
Type	#define Variant 1 Uint16 Variant 2		
Unit	ms		
Range	--	N_Bs value	
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	--	--
	Post Build	L	Variant 2
Scope	module		
Dependency	--		

Name	CANTP_NCS		
Description	Value in ms 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.		
Type	#define Variant 1 Uint16 Variant 2		
Unit	ms		
Range	--	N_Cs value	
Configuration Class	***Pre-compile***	x	Variant 1
	Link time	--	--
	Post Build	L	Variant 2
Scope	module		
Dependency	--		

Name	CANTP_DL		
Description	Data Length Code of this TxNsdu. In case of variable length message, this value indicates the minimum data length.		
Type	#define		
Unit	bytes		
Range	--	DL value	
Configuration Class	***Pre-compile***	x	All Variants
	Link time	--	--
	Post Build	--	--
Scope	module		
Dependency	--		

Name	CANTP_TX_CHANNEL		
Description	Link to the connection channel which has to be used for transmission of this N-PDU.		
Type	#define		
Unit	--		
Range	--		

Configuration Class	***Pre-compile***	x	All Variants
	Link time	--	--
	Post Build	--	--
Scope	module		
Dependency	--		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
N_Sa	0..1	module
N_Ta	0..1	module

10.2.5 N_Ta

SWS Item	**CanTp139:**
Container Name	N_Ta
Description	The following parameters need to be configured for each RxNsdu or TxNsdu with the `CANTP_ADDRESSING_FORMAT` set to `CANTP_EXTENDED`
Configuration Parameters	

Name	`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.		
Type	`#define`		
Unit	`--`		
Range	--	N_Ta value	
Configuration Class	***Pre-compile***	x	All Variants
	Link time	--	--
	Post Build	--	--
Scope	module		
Dependency	--		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
None	--	--

10.2.6 N_Sa

SWS Item	**CanTp168: :**
Container Name	N_Sa
Description	The following parameters need to be configured for each RxNsdu or TxNsdu with the CANTP_ADDRESSING_FORMAT set to CANTP_EXTENDED
Configuration Parameters	

Name	CANTP_NSA		
Description	If an RxNsdu or a TxNsdu is configured for extended addressing format, this parameter contains the transport protocol source address's value.		
Type	#define		
Unit	--		
Range	--	N_Sa value	
Configuration Class	***Pre-compile***	x	All Variants
	Link time	--	--
	Post Build	--	--
Scope	module		
Dependency	--		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
None	--	--

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.

SWS Item	**CanTp140:**	
Information elements		
Information element name	***Type / Range***	***Information element description***
CANTP_VENDOR_ID	#define/ uint16	Vendor ID of the dedicated implementation of this module according to the AUTOSAR vendor list
CANTP_MODULE_ID	#define/ 0x23	Module ID of this module from Module List
CANTP_AR_MAJOR_VERSION	#define/ uint8	Major version number of AUTOSAR specification, which the corresponding implementation is based on.
CANTP_AR_MINOR_VERSION	#define/ uint8	Minor version number of AUTOSAR specification, which the corresponding implementation is based on.
CANTP_AR_PATCH_VERSION	#define/ uint8	Patch level version number of AUTOSAR specification, which the corresponding implementation is based on.
CANTP_SW_MAJOR_VERSION	#define/ uint8	Major version number of the vendor specific implementation of the module. This numbering is vendor specific.
CANTP_SW_MINOR_VERSION	#define/ uint8	Minor version number of the vendor specific implementation of the module. This numbering is vendor specific.
CANTP_SW_PATCH_VERSION	#define/ uint8	Patch level version number of the vendor specific implementation of the module. This numbering is vendor specific.

11 Changes to Release 1

11.1 Deleted SWS Items

SWS Item	*Rationale*
CanTp004	Requirement obsolete
CanTp015	Covered by CanTp001
CanTp026	Requirement obsolete
CanTp036	Requirement obsolete
CanTp037	Requirement obsolete
CanTp038	Requirement obsolete
CanTp039	Requirement obsolete
CanTp097	Multiple connection now is available
CanTp099	Requirement partly wrong

11.2 Replaced SWS Items

SWS Item of Release 1	*replaced by SWS Item*	*Rationale*
CanTp005	CanTp101	Covered by CanTp101
CanTp007	CanTp101	Covered by CanTp101
CanTp025	CanTp140	modification of type
CanTp041	CanTp137	Grouping of requirement and modification of name
CanTp042	CanTp138	Grouping of requirement and modification of name
CanTp043	CanTp137	Grouping of requirement and modification of name
CanTp044	CanTp138	Grouping of requirement and modification of name
CanTp045	CanTp137	Grouping of requirement and modification of name
CanTp046	CanTp138	Grouping of requirement and modification of name
CanTp047	CanTp137	Grouping of requirement and modification of name
CanTp048	CanTp138	Grouping of requirement and modification of name
CanTp049	CanTp137	Grouping of requirement and modification of name
CanTp050	CanTp138	Grouping of requirement and modification of name
CanTp051	CanTp139	Grouping of requirement and modification of name
CanTp052	CanTp139	Grouping of requirement and modification of name
CanTp053	CanTp138	Grouping of requirement and modification of name
CanTp054	CanTp137	Grouping of requirement and modification of name
CanTp056	CanTp138	Grouping of requirement and modification of name
CanTp060	CanTp138	Grouping of requirement and modification of name
CanTp061	CanTp137	Grouping of requirement and modification of name
CanTp062	CanTp138	Grouping of requirement and modification of name
CanTp063	CanTp137	Grouping of requirement and modification of name
CanTp065	CanTp140	Grouping of requirement and modification of name
CanTp066	CanTp140	Grouping of requirement and modification of name
CanTp068	CanTp137	Grouping of requirement and modification of name
CanTp069	CanTp138	Grouping of requirement and modification of name
CanTp108	CanTp137	Grouping of requirement and modification of name
CanTp109	CanTp138	Grouping of requirement and modification of name
CanTp113	CanTp137	Grouping of requirement and modification of name

11.3 Changed SWS Items

SWS Item	*Rationale*
CanTp001	File structure deleted and reported to CanTp156
CanTp010	Modification of the figure : addition of CanTp_Shutdown call in CANTP_OFF
CanTp019	Addition of Range value for parameter Change CanTpRxPduPtr parameter to const
CanTp020	Addition of Range value for parameter
CanTp032	Addition of ISO 15765-4 reference
CanTp040	Change of parameter name
CanTp074	Requirement that CanIf confirms only successful transmission confirmation
CanTp075	Requirement for the CAN Tp behavior in case of transmission time-out
CanTp082	Requirement when the CAN Tp has to call again for a buffer to be provided
CanTp096	Addition of several connection management
CanTp114	Requirement that it is a pre-compile time configuration

11.4 Added SWS Items

SWS Item	*Rationale*
CanTp156	File structure from CanTp001 reworked in this requirement
CanTp119	ISO 15765-4
CanTp120	Concurrent connections
CanTp121	Concurrent connections
CanTp122	Concurrent connections
CanTp123	Concurrent connections
CanTp124	Concurrent connections
CanTp130	Dem.h file inclusion
CanTp132	*CANTP_DEV_ERROR_DETECT* switch
CanTp133	Production error detection
CanTp134	Development Error Tracer
CanTp137	RxNsDu Configuration
CanTp138	RxNsDu Configuration
CanTp139	N_Ta Configuration
CanTp140	Published information Configuration
CanTp146	Configuration tool
CanTp147	Configuration tool
CanTp150	Design rules
CanTp151	Design rules
CanTp152	Design rules
CanTp153	Design rules
CanTp155	Design rules
CanTp158	Design rules
CanTp156	File structure
CanTp157	CanTp.h file
CanTp158	Can Tp source code
CanTp159	Configuration files
CanTp160	Reference to c-configuration parameters
CanTp161	Initialization status variable
CanTp162	CanTp_GetVersionInfo Api
CanTp163	CanTp_GetVersionInfo Api
CanTp164	MainFunction Api