# Technical Reference

Version 1.18.01

| | |
|---|---|
| Authors | Peter Herrmann |
| Status | Released |

| | | | |
|---|---|---|---|
| Peter Herrmann | 2006-12-07 | 1.0 | Initial version |
| Peter Herrmann | 2007-06-19 | 1.3 | Update to AUTOSAR Release 2.1.0 |
| Peter Herrmann | 2008-01-18 | 1.4 | Added database  attributes |
| Peter Herrmann | 2008-01-22 | 1.5 | Added pre-compile macros for Tx Confirmation and Rx Indication callbacks |
| Peter Herrmann | 2008-04-07 | 1.6 | Adaptation to MICROSAR document template. |
| Peter Herrmann | 2008-07-31 | 1.7 | Added description for optimizations:<br>- dynamic channel assignment (DYN_CHANNEL_ASSIGNMENT),<br>- single connection (SINGLE_CONN_OPTIMIZED)<br>- single connection pre-compile (SINGLE_CONN_NOPB_OPTIMIZED)<br>- addressing types (STANDARD_ADDRESSING, EXTENDED_ADDRESSING)<br>- partial buffer provision (RX/TX_FULL_BUFFER_PROVISION) |
| Peter Herrmann | 2008-09-16 | 1.8 | Burst transmission |
| Peter Herrmann | 2008-10-30 | 1.9 | Version number updated |
| Peter Herrmann | 2008-11-26 | 1.10 | Single connection optimization |
| Peter Herrmann | 2009-02-20 | 1.11 | MainFunction Rx/Tx split. Additional user NSDU-ID (remapping). Additional SetSTmin, SetBS API. |
| Peter Herrmann | 2009-06-10 | 1.11.82 | Conversion to MICROSAR Technical Reference.<br>Added Multiple Configuration, Cancel Transmit Request. |
| Peter Herrmann | 2009-07-01 | 1.12.00 | Mixed-11 addressing added.<br>AUTOSAR Release 4 PduR API callout functions added. |
| Peter Herrmann | 2009-29-09 | 1.13.00 | Corrected chapter 7.2.4 (Timing limitations for N_Br, N_Cs). |
| Peter Herrmann | 2009-11-04 | 1.14.00 | Added non AUTOSAR Spec. conform special feature to avoid all Pre-Pass indications to the DEM (see 7.1.9).<br>Added special feature to accelerate the FF routing (see 7.1.10). |

| | | | |
|---|---|---|---|
| Peter Herrmann | 2010-02-18 | 1.15.00 | Additional AUTOSAR 4.x API: “CanTp_ChangeParameterRequest” and belonging callout function “PduR_CanTpChangeParameterConfirmation”.<br>Additional Vector specified API “CanTp_ReadParameterRequest”.<br>Added Callout description for “EcuM_GeneratorCompatibilityError”.<br>Added description for the mapping of critical sections.<br>Deleted obsolete CanTp_DlcErrorNotification callout function from CanIf |
| Peter Herrmann | 2010-08-13 | 1.16.00 | Additional AUTOSAR 4.x API: “CanTp_CancelTransmitRequest” and “CanTp_CancelReceiveRequest”.<br>Added description of postbuild parameters CanTpMaxNum(Rx/Tx)Sdus, CanTpMaxNum(Rx/Tx)Channels_PbLimit. |
| Peter Herrmann | 2010-11-03 | 1.17.00 | Additional description for callout functions with possibly disabled interrupts.<br>Deleted API function PduR_CanTpGetAvailableTxBuffer which is obsolete with AR 4.0 |
| Peter Herrmann | 2011-02-25 | 1.18.00 | Added further (AR4 and customer specific) extensions |
| Peter Herrmann | 2011-03-30 | 1.18.01 | After logging and rework |

|  |  |  |
|---|---|---|
| [1] | AUTOSAR_SWS_CANTransportLayer.pdf | 3.0.0 |
| [2] | AUTOSAR_SWS_CANInterface.pdf | 4.0.0 |
| [3] | AUTOSAR_SWS_PDURouter.pdf | 3.0.0 |
| [4] | /ISO/TF2/: ISO FDIS 15765-2; Road vehicles — Diagnostics on CAN — Part 2: Network layer services | 2009-09-06 |
| [5] | AUTOSAR_SWS_DiagnosticEventManager.pdf | 4.0.0 |
| [6] | AUTOSAR_SWS_DevelopmentErrorTracer.pdf | 3.0.0 |
| [7] | AUTOSAR_SRS_BSWGeneral.pdf | 3.0.0 |
| [8] | AUTOSAR_TR_BSWModuleList.pdf | 1.4.0 |
| [9] | Application Note AN-ISC-8-1118 – MICROSAR BSW Compatibility Check |  |
| [10] | AUTOSAR_SWS_CAN_TP.pdf | 2.3.0 |

|  |  |
|---|---|
| ⚠ | We have configured the programs in accordance with your specifications in the questionnaire. Whereas the programs do support other configurations than the one specified in your questionnaire, Vector´s release of the programs delivered to your company is expressly restricted to the configuration you have specified in the questionnaire. |

The component history gives an overview over the important milestones that are supported in the different versions of the component.

For details please see “Document Change History” within [1] and [10].

This document describes the functionality, API and configuration of the AUTOSAR BSW module CANTP as specified in [1] and [10].

| | | |
|---|---|---|
| | 3 and 4 (configurable) | |
| | pre-compile, post-build | |
| | CANTP_VENDOR_ID | 30 decimal<br>(= Vector-Informatik, according to HIS) |
| | CANTP_MODULE_ID | 35 decimal<br>(according to ref. [8]) |

.

According to AUTOSAR basic software architecture, CANTP provides services for

Segmentation of data in transmit direction

Reassembly of data in receive direction

Control of data flow

Detection of errors in segmentation sessions

AUTOSAR module specifications are based on existing standards. Thus this AUTOSAR CAN Transport Layer specification is based on the international standard ISO 15765 which is the most widespread standard in the automotive domain.

ISO 15765 contains four sections and describes two applicable CAN Transport Layer specifications: ISO 15765-2 for OEM enhanced diagnostics (see [4]) and ISO 15765-4 for OBD diagnostics. Concerning the transport layer, ISO 15765-4 (the section of ISO 15765 which also covers the data link layer and the 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 the CAN transport protocol is mainly used for vehicle diagnostic systems, its design also incorporates requirements from other CAN based systems employing transport-layer protocols.

The following figure shows where the CANTP is located in the AUTOSAR architecture.

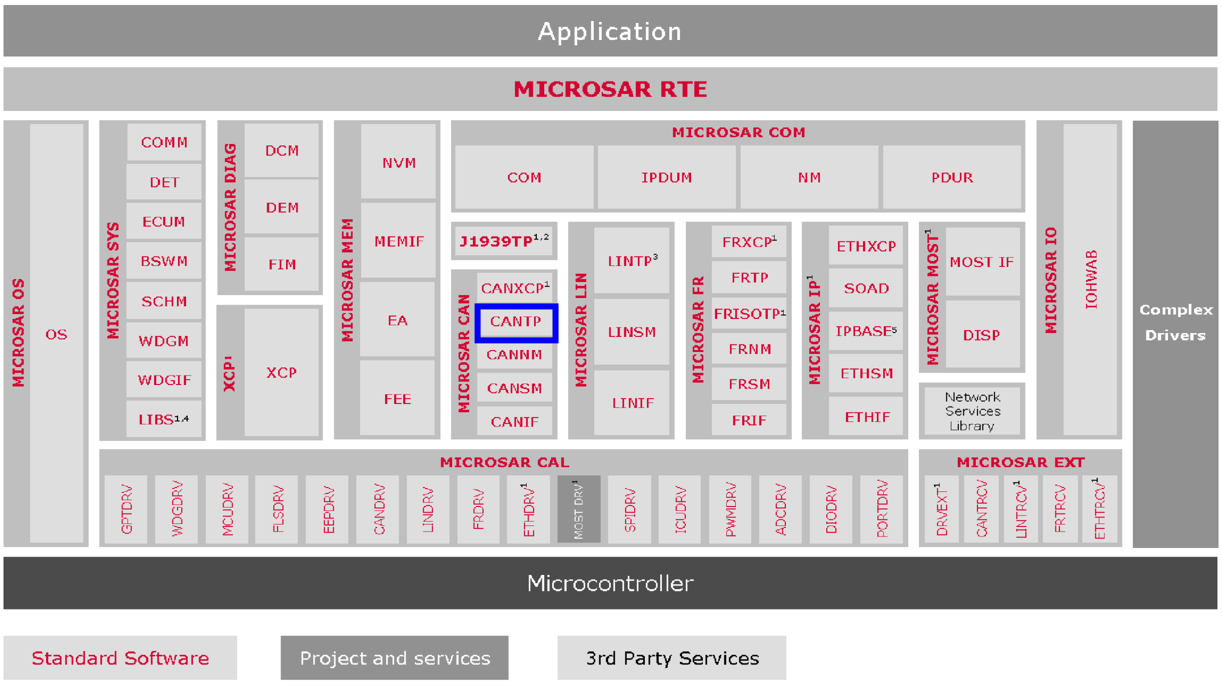


Figure 2-1 AUTOSAR architecture

The upper interface of CANTP provides a PDUR module global access for transmitting and receiving data.

This access is realized via a CAN N-SDU identifier (CAN NSduId). The 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 the implementation.

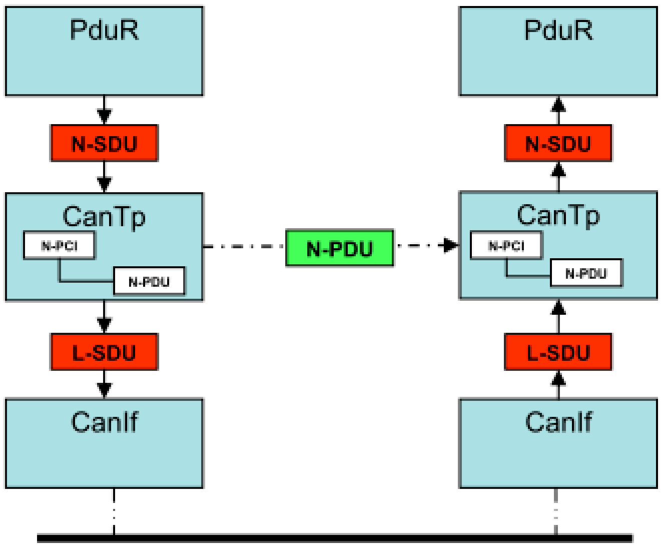


Figure 2-2 CAN Transport Layer interactions

The next figure shows the interfaces to adjacent modules of the CANTP. These interfaces are described in chapter 5.

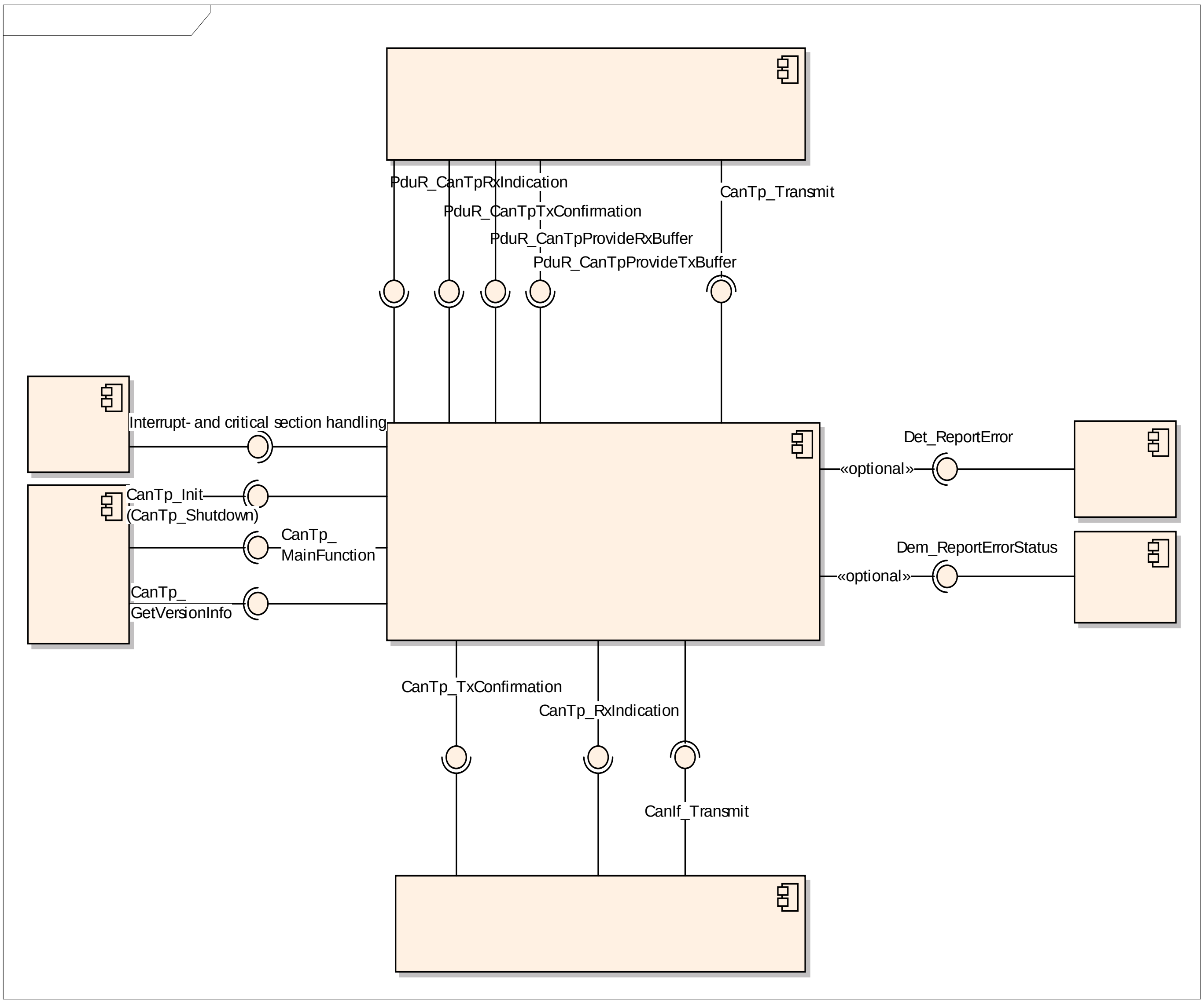


Figure 2-3 Interfaces to adjacent modules of the CANTP

The buffer provision calls from AR3
PduR_CanTpProvideRxBuffer and
PduR_CanTpProvideTxBuffer
have been changed by AUTOSAR to
PduR_CanTpStartOfReception, PduR_CanTpCopyRxData and
PduR_CanTpCopyTxData
within AR4.

In addition to the existing CanIf_Transmit API a cancellation API function is supported since AR4. The CanIf_CancelTransmit shall allow to cancel transmit requests which are not already handed physically to the CAN bus.

The implemented features cover the complete functionality specified in [1] and [2] except the entries in “Table 3-2 Not supported SWS features” below.

Extensions and limitations concerning the AUTOSAR features are presented in chapter 7 in more detail.

The following additional features are supported:

| |
|---|
| Burst transmission |
| Full Duplex (in addition to [10]) |
| CANTP initialization (in addition to [10]) |
| Padding pattern |
| Splitted CanTp_MainFunction |
| Dynamic Flow Control content (in addition to [10]) |

Table 3-1 Supported additional features

The following features described in [1] are not supported:

| |
|---|
| Half Duplex |
| Shared channels |

Table 3-2 Not supported SWS features

After power up, CANTP is in a state called CANTP_OFF. In this state, the CANTP is not yet initialized and therefore may not perform any communication task.

This service uses the given configuration set to initialize all global variables of the CAN Transport Layer and brings it to the idle state (state = CANTP_ON, but neither transmission nor reception are possible yet).

The CANTP module has two internal states, `CANTP_OFF` and `CANTP_ON`. After power up the CANTP is in the `CANTP_OFF` state.

The CANTP changes to the internal state `CANTP_ON` when it has been successfully nitialized with `CanTp_Init()`.

The CANTP performs segmentation and reassembly tasks only when it is in the `CANTP_ON` state. The function `CanTp_Init` initializes 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`).

If called when the CANTP module is in the global state `CANTP_ON`, the function `CanTp_Init` returns the module to state Idle (state = CANTP_ON, but neither transmission nor reception are in progress) and the module looses all current connections.

The function `CanTp_Shutdown` stops the CANTP module properly.

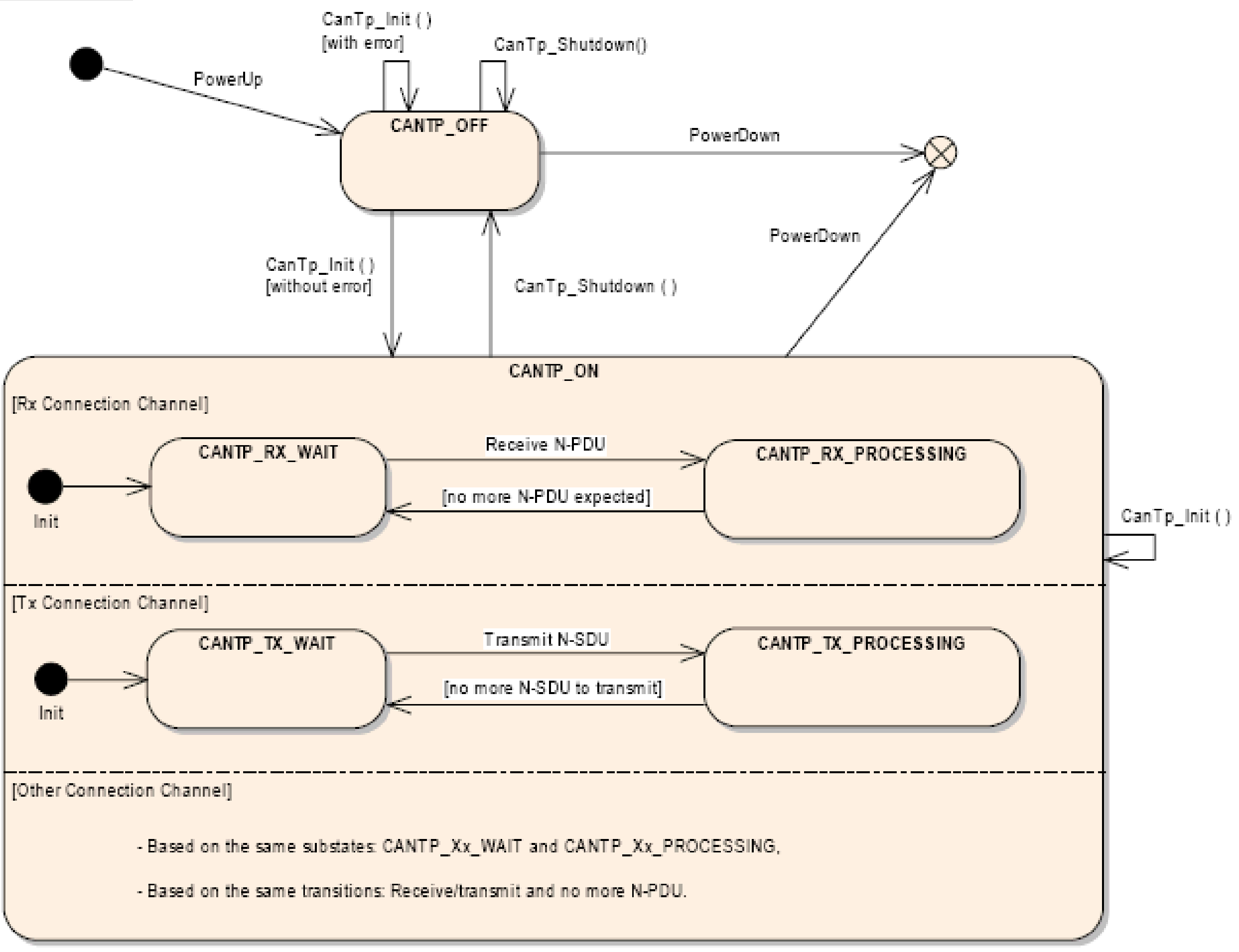


Figure 3-1 CANTP internal states

| | |
|---|---|
| () | Overall module initialization prior to any usage. |
| | Obtains the version information of the CANTP module. |
| | Closes all connections and sets the CANTP module into CANTP_OFF state. |
| | Request the transfer of segmented data. |
| | Entry point for scheduling the CANTP module. |

This is the main function serving as time-basis for scheduling the CANTP (entry-point for scheduling).

Please note, that the configured task cycle is used to calculate any kind of timing behavior (e.g. timeouts or delays.). So this function must be called periodically with the 'CanTpMainFunctionPeriod' which is actually configured in the generation tool.

To optimize the communication stack, AUTOSAR limits the CANTP buffering capacity. Therefore, the CANTP copies N-SDU payload directly from an upper layer (e.g. DCM or PDU Router) to the CAN driver and vice-versa.

Thus, in order to guarantee data consistency, the upper layer assure the following rules (see [1] 5.1.4):

At transmission time: the N-SDU data payload remains unchanged from the transmit request until the transmit confirmation has been received or the next buffer is requested from the CANTP.

At reception time: the N-SDU data access is locked from the buffer allocation request until the reception indication or the next buffer allocation request has been received.

This means that the application is responsible for data consistency during transmission and reception!

For more information on these functions, refer to the PDU Router module specification [3].

| | |
|---|---|
| ⚠ | The buffer provision callback functions are called directly from the CANTP calling context. This can also be on interrupt level. So the callback functions must assure that their duration is strictly limited to avoid the loss of information due to a long interrupt disable period. |

Different messages from different busses (this can be other CAN buses or, in the case of a gateway ECU, also other  buses like FlexRay, LIN etc.) have to be handled at the same time by the CANTP as well as transmission requests from the local ECU. This handling can be accomplished as well from interrupt as also from task level. Thus a synchronization mechanism is implemented to guarantee data consistency amongst all connections.

The synchronization mechanism defined by AUTOSAR covers the entering and leaving of so called critical sections. Different critical sections can be handled by using different so called “Exclusive Areas”.

CANTP uses the critical sections CANTP_EXCLUSIVE_AREA_0 and CANTP_EXCLUSIVE_AREA_1. These critical sections can be entered from task and interrupt context depending on the configuration of the calling layer.

The implementation of the critical sections must avoid that multiple relevant tasks or interrupt service routines can enter any of the critical sections at the same time, i.e. they may not interrupt each other.

Relevant interrupts in the context of the CANTP are interrupts originated from physical bus events (CAN, LIN, FlexRay etc.).

Relevant tasks in the CANTP context are all tasks which call CANTP API functions. Usually these tasks are limited to tasks on which other BSW modules (DCM, COM, ECUM) are mapped to.

E.g.: For an implementation of the critical sections it could be sufficient to

Disable interrupt levels up to the highest bus interrupt level (interrupt controller with interrupt levels supposed) assuming that the task rescheduling is thereby inhibited too

Disable all bus relevant interrupts if all modules calling CANTP API functions are mapped to one task (e.g. SchM task) or a non-preempitve OS is used

The difference between CANTP_EXCLUSIVE_AREA_0 and CANTP_EXCLUSIVE_AREA_1 is the frequency of enter/leave calls.

CANTP_EXCLUSIVE_AREA_1 is used in CanTp_MainFunctionRx and CanTp_MainFunctionTx while iterating through all connections doing one enter/leave per iteration. It is possible to implement this critical section with higher performance mechanisms (e.g., global interrupt lock) if necessary. This is typically used in gateway applications with configurations containing many connections on several physical busses.

CANTP_EXCLUSIVE_AREA_0 and CANTP_EXCLUSIVE_AREA_1 are mutually exclusive. The implementation must ensure that only one of the critical sections can be entered at the same time.

Please note that these are only examples and that the actual implementation of the critical sections is highly dependent on the platform architecture and the system configuration.

By default, development errors are reported to the DET using the service `Det_ReportError()` as specified in [6], if development error reporting is enabled (i.e. pre-compile parameter `CANTP_DEV_ERROR_DETECT==STD_ON)`.

If another module is used for development error reporting, the function prototype for reporting the error can be configured by the integrator, but must have the same signature as the service `Det_ReportError()`.

The reported CANTP ID is 35.

The reported service IDs identify the services which are described in 5.1. The following table presents the service IDs and the related services:

| | |
|---|---|
| 0x01 | CanTp_Init |
| 0x02 | CanTp_Shutdown |
| 0x03 | CanTp_Transmit |
| 0x04 | CanTp_RxIndication |
| 0x05 | CanTp_TxConfirmation |
| 0x06 | CanTp_MainFunction |
| 0x07 | CanTp_GetVersionInfo |

Table 3-3 Mapping of service IDs to services

The errors reported to DET are described in the following table:

| | | |
|---|---|---|
| 0x01 | CANTP_E_PARAM_CONFIG | API called with wrong parameters |
| 0x02 | CANTP_E_PARAM_ID | API called with wrong parameters |
| 0x04 | CANTP_E_PARAM_ADDRESS | API called with wrong parameters |
| 0x20 | CANTP_E_UNINIT | API service used without module initialization |
| 0x30 | CANTP_E_INVALID_TX_ID | Invalid Transmit PDU identifier |
| 0x40 | CANTP_E_INVALID_RX_ID | Invalid Receive PDU identifier |
| 0x50 | CANTP_E_INVALID_TX_BUFFER | Invalid Transmit buffer address |
| 0x60 | CANTP_E_INVALID_RX_BUFFER | Invalid Receive buffer address |
| 0x70 | CANTP_E_INVALID_TX_LENGTH | Invalid data length of the transmit PDU |
| 0x80 | CANTP_E_INVALID_RX_LENGTH | Invalid data length of the receive PDU |
| 0x90 | CANTP_E_INVALID_TA_TYPE | Functional addressing Tx I-Pdu message does not fit into a SF |
| 0x5C | NTFRSLT_E_CANTP_INVALID_STATE | This error is used if the internal state machine of the CANTP shows an invalid status. |

Table 3-4 Errors reported to DET

By default, production code related errors are reported to the DEM using the service `Dem_ReportErrorStatus()` as specified in [5], if production error reporting is enabled (i.e. pre-compile parameter `CANTP_PROD_ERROR_DETECT==STD_ON`).

If another module is used for production code error reporting, the function prototype for reporting the error can be configured by the integrator, but must have the same signature as the service `Dem_ReportErrorStatus()`.

The errors reported to DEM are described in the following table:

| | |
|---|---|
| CANTP_E_OPER_NOT_SUPPORTED | Requested operation is not supported |
| CANTP_E_RX_COM | Event reported on completion of a reception operation (only AUTOSAR 4) |
| CANTP_E_TX_COM | Event reported on completion of a transmission operation (only AUTOSAR 4) |
| CANTP_E_COM<br>(named CANTP_E_COMM within AUTOSAR 3.1) | Another error occurred during a reception or a transmission (AUTOSAR 3 and AUTOSAR4) |

Table 3-5 Errors reported to DEM

This chapter gives necessary information for the integration of the MICROSAR CANTP into an application environment of an ECU.

The delivery of the CANTP contains the files which are described in the chapters 4.1.1 and 4.1.2:

| | |
|---|---|
| CanTp.c | Source file of the CANTP. |
| CanTp.h | Header file of the CANTP. |
| CanTp_Hooks.h | Header  file of the monitoring hook function definitions (included from CanTp.c). |
| CanTp_Cbk.h | Header file of the CANTP callback functions. |
| CanTp_Types.h | Header file of the CANTP type definitions. |
| CanTp_Priv.h | Header file of the CANTP extended type definitions (Vector extensions to AUTOSAR). |
| CanTp.lib | Library of the CANTP |

Table 4-1 Static files

Additional hook functions can be provided by defining the function declarations supported within the CanTp_Hooks.h header file.

The CanTp callback functions

- CanTp_RxIndication, called at the reception of each CAN frame and
- CanTp_TxConfirmation, called at the transmit confirmation of each CAN frame

are supplied with the following tags at the beginning and the end of the function:

```
# define CanTpHook_OnBegin_CanTp_RxIndication()
# define CanTpHook_OnEnd_CanTp_RxIndication()
# define CanTpHook_OnBegin_CanTp_TxConfirmation()
# define CanTpHook_OnEnd_CanTp_TxConfirmation().
```

Please note that the provided hook functions are all of the type “void func(void)”.

The dynamic files are generated by the configuration tool [config tool].

| | |
|---|---|
| CanTp_Cfg.h | This is the generated header file of CANTP containing pre-compile switches and providing symbolic defines. |
| CanTp_Cfg.c | This is the generated source file of CANTP containing pre-compile-time configurable parameters |
| CanTp_Lcfg.h | This is the generated header file of CANTP containing link-time configurable symbols |
| CanTp_Lcfg.c | This is the generated source file of CANTP containing link-time configurable parameters |
| CanTp_PBcfg.c | This is the generated source file of CANTP containing post-build-time configurable parameters |
| CanTp_PBcfg.c | This is the generated source file of CANTP containing post-build-time configurable parameters |

Table 4-2 Generated files

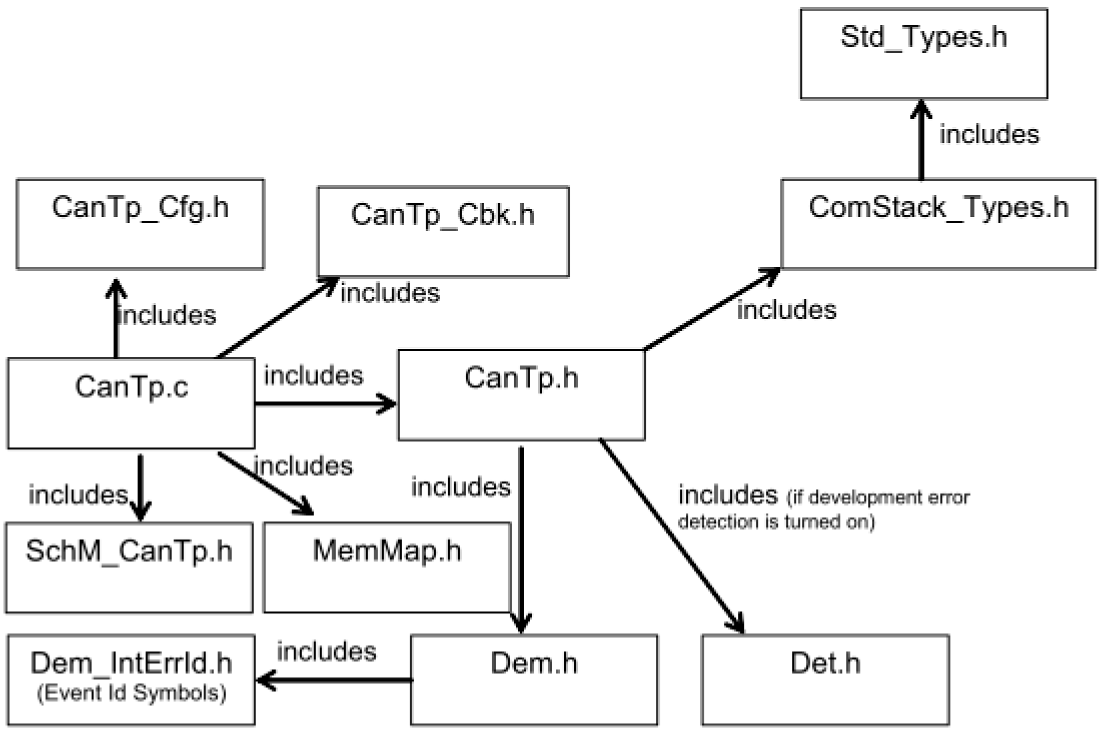


Figure 4-1 Include structure

The objects (e.g. variables, functions, constants) are declared by compiler independent definitions – the compiler abstraction definitions. Each compiler abstraction definition is assigned to a memory section.

The following table contains the memory section names and the compiler abstraction definitions defined for the CANTP and illustrates their assignment among each other.

| | CANTP_CONST | CANTP_PBCFG | CANTP_CODE | CANTP_APPL_CODE | CANTP_APPL_DATA | CANTP_VAR_NOINIT | CANTP_VAR |
|---|---|---|---|---|---|---|---|
| CANTP_START_SEC_CODE<br>CANTP_STOP_SEC_CODE | | | ■ | | | | |
| CANTP_START_SEC_VAR_NOINIT_UNSPECIFIED<br>CANTP_START_SEC_VAR_NOINIT_UNSPECIFIED | ■ | | | | | ■ | |
| CANTP_START_SEC_CONST_16BIT<br>CANTP_STOP_SEC_CONST_16BIT | ■ | | | | | | |
| CANTP_START_SEC_CONST_32BIT<br>CANTP_STOP_SEC_CONST_32BIT | ■ | | | | | | |
| CANTP_START_SEC_CONST_PBCFG_ROOT<br>CANTP_START_SEC_CONST_PBCFG_ROOT | | ■ | | | | | |
| CANTP_START_SEC_PBCFG_ROOT<br>CANTP_START_SEC_PBCFG_ROOT | | ■ | | | | | |
| CANTP_START_SEC_VAR_NOINIT_8BIT<br>CANTP_START_SEC_VAR_NOINIT_8BIT | | | | | | ■ | |

Table 4-3 Compiler abstraction and memory mapping

For an interfaces overview please see Figure 2-3.

The CANTP API consists of services, which are realized by function calls.

| | |
|---|---|
| | |
| `void ( void )` | |
| | |
| void | -- |
| | |
| void | -- |
| | |
| This service uses the given configuration set to initialize all global variables of the CANTP and sets it in the idle state (state = CANTP_ON). No transmissions or receptions are currently pending.<br>The CANTP is not configured and must not perform any communication task unless this initialization function was called at least once. | |
| | |
| Only available if enabled at compile time (`CANTP_HAVE_INIT_CFG_PTR == STD_OFF`) | |
| Expected Caller Context | |
| Called from the EcuManager during system startup. | |

Table 5-1 CanTp_Init: until AR 3 (without ConfigPointer)

| | |
|---|---|
| `void ( const void* const pCfgPtr )` | |
| | |
| pCfgPtr | Pointer to a given configuration set. |
| | |
| void | -- |
| | |
| This service uses the configuration set addressed to by the pCfgPtr to initialize all global variables of the CANTP and sets it in the idle state (state = CANTP_ON). No transmissions or receptions are currently pending.<br>The CANTP is not configured and must not perform any communication task unless this initialization function was called at least once. | |
| | |
| Only available if enabled at compile time (CANTP_HAVE_INIT_CFG_PTR == STD_ON)<br>If multiple configurations are used then this pointer is used to identify the appropriate identity. | |
| Expected Caller Context | |
| Called from the EcuManager during system startup. | |

Table 5-2 CanTp_Init: with ConfigPointer (Vector extension to AR 3)

| | |
|---|---|
| `void` | `( const CanTp_ConfigType* pCfgPtr )` |
| | |
| pCfgPtr | Pointer to a given configuration set. |
| | |
| void | -- |
| | |
| This service uses the configuration set addressed to by the pCfgPtr to initialize all global variables of the CANTP and sets it in the idle state (state = CANTP_ON). No transmissions or receptions are currently pending.<br>The CANTP is not configured and must not perform any communication task unless this initialization function was called at least once. | |
| | |
| If multiple configurations are used then this pointer is used to identify the appropriate identity. | |
| Expected Caller Context | |
| Called from the EcuManager during system startup. | |

Table 5-3 CanTp_Init: with ConfigPointer

| | |
|---|---|
| `void ( Std_VersionInfoType* const pVersionInfo )` | |
| | |
| pVersionInfo | The version information of the CANTP module is stored in the structure referenced by this pointer. |
| | |
| void | -- |
| | |
| This service returns the version information of this module.<br>The version information includes: Module Id, Vendor Id and Vendor specific version numbers. | |
| | |
| Only available if enabled at compile time (CANTP_VERSION_INFO_API == STD_ON) | |
| Expected Caller Context | |
| Interrupt or task context | |

Table 5-4 CanTp_GetVersionInfo

| | |
|---|---|
| `void` | `( void )` |
| | |
| Void | -- |
| | |
| Void | -- |
| | |
| This service closes all pending transport protocol connections, frees all resources and sets the corresponding CANTP module into the CANTP_OFF state.<br>No further communication is possible unless (re-)initialization via CanTp_Init(). | |
| | |
| No notification about eventually pending frame transmissions or receptions will be raised. | |
| Expected Caller Context | |
| Interrupt or task context | |

Table 5-5 CanTp_Shutdown

| | |
|---|---|
| Std_ReturnType (const PduIdType CanTpTxSduId,<br>const PduInfoType*, const CanTpTxInfoPtr<br>) | |
| | |
| CanTpTxSduId | Unique CANTP module identifier of the CAN N-SDU to be transmitted.<br>Range: 0..(maximum number of I-PDU IDs) –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.<br><br>Basic information about a PDU of any type:<br>typedef struct<br>{<br>  uint8          *SduDataPtr;<br>  PduLengthType SduLength;<br>} PduInfoType; |
| | |
| E_OK | Request started successfully. |
| E_NOT_OK | The request cannot be started (e.g. a transmit request is in progress with the same N-SDU identifier). |
| | |
| This service is used to request the transfer of segmented data.<br>If data length is less than 7 or 6,  depending on the addressing format (normal, extended. mixed11), a SF N-PDU is sent. However, if data length is greater than 7 or 6, a multiple frame transmission session is initiated.<br>When the transmit request has been completed, the CANTP notifies the upper layer by calling the PduR_CanTpTxConfirmation callback function.<br>Also, if an error occurred (overflow, timeout etc.), the transmit request is aborted and the PduR_CanTpTxConfirmation callback is called with the appropriate error result value. | |
| | |
| 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 to be transmitted. To access the Tx buffer, the CANTP calls the PduR_CanTpProvideTxBuffer service.<br>If the CanTp_Transmit service is called for a N-SDU identifier which is being used in a currently running CanTp session, the CanTp rejects the request.<br>To guarantee data consistency, the upper layer (e.g. DCM, PduRouter or AUTOSAR COM) must lock the transmit data area until the confirmation notification occurs.<br>In case of a functional connection and a data length that does not fit into one single frame, the transmit request is cancelled and an application notification (NTFRSLT_E_UNEXP_PDU) is issued. | |

| Expected Caller Context |
|---|
| Either task context (e.g. if called from other BSW modules like DCM, COM etc.) or interrupt context (e.g. if called from the PduR in case of a gateway configuration) |

Table 5-6 CanTp_Transmit

| | |
|---|---|
| `Std_ReturnType` | `(PduIdType CanTpTxSduId)` |
| | |
| CanTpTxSduId | Unique CANTP module identifier of the CAN N-SDU to be cancelled for transmission.<br>Range: 0..(maximum number of I-PDU IDs) –1 |
| | |
| E_OK<br>E_NOT_OK | Cancellation request accepted,<br>Cancellation request rejected.<br>The reason can be that the request is issued for an N-SDU that is not segmented or the request is issued after the last CF has been requested for transmission or a cancellation is not possible for the related N-SDU due to configuration. |
| | |
| This service is used to cancel the transmission of an ongoing N-SDU.<br>This is only performed for segmented data but not for Single Frames.<br>When the cancel transmit request has been completed, the CANTP notifies the upper layer by calling the PduR_CanTpTxConfirmation callback function with the ‘NTFRSLT_E_CANCELATION_OK‘ notification result and returns with E_OK.<br>If an error occurred, the cancel transmit request is aborted and E_NOT_OK is returned.<br>If the parameter value is invalid, the development error CANTP_E_PARAM_ID is raised and E_NOT_OK returned.<br>If the parameter value indicates a cancel transmission request for an N-SDU that is not on transmission process the DEM with EventId = CANTP_E_OPER_NOT_SUPPORTED and EventStatus = DEM_EVENT_STATUS_PREFAILED is called and E_NOT_OK is returned. | |
| | |
| If the CanTp_CancelTransmitRequest service is called for a N-SDU identifier which is undefined then the CanTp rejects the request.<br>It is not guaranteed that a frame just being transmitted can be stopped. | |
| | |
| Task context | |

Table 5-7 CanTp_CancelTransmitRequest

If the cancellation in hardware is not supported then the resulting confirmations, given to the CanTp unexpectedly (due to sent messages which were expected to be cancelled) can corrupt the proper CanTp handling for the concerning connection. This corruption could be avoided by delaying further transmit actions on the same connection after a cancellation until the appearance of unexpected confirmations can definitely be excluded.

| | |
|---|---|
| | |
| `Std_ReturnType (PduIdType CanTpRxSduId)` | |
| | |
| CanTpRxSduId | Unique CANTP module identifier of the CAN N-SDU to be received.<br>Range: 0..(maximum number of I-PDU IDs) –1 |
| | |
| E_OK | Cancellation request is accepted, |
| E_NOT_OK | Cancellation request is rejected.<br>The reason can be that the request is issued for an N-SDU that is not segmented or the request is issued for an N-SDU that is not in the process of reception. |
| | |
| This service is used to cancel the reception of an ongoing N-SDU.<br>If the CanTp_CancelReceiveRequest service has been successfully executed the CanTp calls the PduR_CanTpRxIndication with the notification result ‘NTFRSLT_E_CANCELATION_OK’.<br>If development error detection is enabled the function CanTp_CancelReceiveRequest checks the validity of CanTpRxSduId parameter. If the parameter value is invalid, the CanTp_CancelReceiveRequest function raises the development error CANTP_E_PARAM_ID and returns E_NOT_OK.<br>If the parameter value indicates a cancel reception request for an N-SDU that it is not on reception process the DEM is called with EventId = CANTP_E_OPER_NOT_SUPPORTED and EventStatus = DEM_EVENT_STATUS_PREFAILED and E_NOT_OK is returned. | |
| | |
| The CanTp rejects the request for receive cancellation in case of a Single Frame reception or if the CanTp is in the process of receiving the last Consecutive Frame of the N-SDU (i.e. the service is called after N_Cr timeout is started for the last Consecutive Frame). In this case the CanTp returns E_NOT_OK. | |
| | |
| Task context | |

Table 5-8 CanTp_CancelReceiveRequest

| | |
|---|---|
| `void ( void )` | |
| | |
| Void | -- |
| | |
| Void | -- |
| | |
| Entry point for scheduling the CANTP module. This function must be called periodically with the `CanTpMainFunctionPeriod` configured in the generation tool. | |
| | |
| -- | |
| Expected Caller Context | |
| task context | |

Table 5-9 CanTp_MainFunction

| | |
|---|---|
| | |
| `void ( void )` | |
| | |
| Void | -- |
| | |
| Void | -- |
| | |
| Entry point for scheduling the CANTP module on the reception side. This function must be called periodically with the ‘CanTpMainFunctionPeriod’ configured in the generation tool.<br>For an optimal calling sequence the CanTp_MainFunctionRx is typically called directly before the diagnostic application is called to provide the incoming request within the same task cycle. | |
| | |
| The split of the CanTp_Mainfunction is optional and can be configured in the generation tool (CANTP_RXTX_MAINFUNCTION_API == STD_ON/STD_OFF). The default is STD_OFF. | |
| Expected Caller Context | |
| task context | |

Table 5-10 CanTp_MainFunctionRx

| | |
|---|---|
| | |
| `void ( void )` | |
| | |
| Void | -- |
| | |
| Void | -- |
| | |
| Entry point for scheduling the CANTP module on the transmission side. This function must be called periodically with the ‘CanTpMainFunctionPeriod’ configured in the generation tool.<br>For an optimal calling sequence the CanTp_MainFunctionTx is typically called after the application (typically the diagnostic module) has processed an incoming request. This sequence supports the transmission of the resulting response as fast as possible. | |
| | |
| The split of the CanTp_Mainfunction is optional and can be configured in the generation tool (CANTP_RXTX_MAINFUNCTION_API == STD_ON/STD_OFF). The default is STD_OFF. | |
| Expected Caller Context | |
| task context | |

Table 5-11 CanTp_MainFunctionTx

| | |
|---|---|
| | |
| `uint8 ( PduIdType CanTpRxSduId, uint8 CanTpSTmin )` | |
| | |
| CanTpRxSduId | ID of CAN N-SDU for which the new value shall be set |
| CanTpSTmin | new value for STmin |
| | |
| STmin | Either the new (valid) value or, in case of an invalid new value the previously set value. |
| | |
| This service is used to set a new value for STmin for the given NSdu-Id. The new value is used for the FCs to be transmitted for each newly started reception until ECU reset. After power on the statically configured default values are restored and used again until a new dynamic change appears. | |
| | |
| The dynamic setting of the STmin time is optional and can be configured in the generation tool (CANTP_ENABLE_EXT_API_STMIN == STD_ON/STD_OFF). The default is STD_OFF.<br>If the "NSdu-Id Remapping" - feature (see 6.2.1.1) is enabled then the 'CanTpRxSduId' parameter must contain the NSdu-Id remapped by the user. Otherwise the internally generated NSdu-Id number is used here (this internally generated NSdu-Id number is provided to the user as a pre-processor constant within the generated CanTp_Cfg.h file). | |
| Expected Caller Context | |
| Interrupt or task context | |

Table 5-12 CanTp_SetSTmin

| | |
|---|---|
| `uint8 ( PduIdType CanTpRxSduId, uint8 CanTpBS )` | |
| | |
| CanTpRxSduId | ID of CAN N-SDU for which the new value shall be set |
| CanTpBS | new value for BS |
| | |
| BS | Either the new (valid) value or, in case of an invalid new value the previously set value. |
| | |
| This service is used to set a new value for BS for the given NSdu-Id. The new value is used for the FCs to be transmitted for each newly started reception until ECU reset. After power on the statically configured default values are restored and used again until a new dynamic change appears. | |
| | |
| The dynamic setting of the blocksize is optional and can be configured in the generation tool (CANTP_ENABLE_EXT_API_BS == STD_ON/STD_OFF). The default is STD_OFF. If the “NSdu-Id Remapping” - feature (see 6.2.1.1) is enabled then the 'CanTpRxSduId' parameter must contain the NSdu-Id remapped by the user. Otherwise the internally generated NSdu-Id number is used here (this internally generated NSdu-Id number is provided to the user as a pre-processor constant within the generated CanTp_Cfg.h file). | |
| Expected Caller Context | |
| Interrupt or task context | |

Table 5-13 CanTp_SetBS

| | |
|---|---|
| `uint8` | `( PduIdType CanTpRxSduId,`<br>`TPParameterType Parameter,`<br>`uint16 Value )` |
| | |
| CanTpRxSduId | ID of CAN N-SDU for which the new value shall be set |
| Parameter | The parameter (either STmin or BS) for which a new value shall be set |
| Value | The new value for the selected 'Parameter' |
| | |
| E_OK<br>E_NOT_OK | If the value could be changed.<br>If the value could not be changed.<br>Possible errors are: Either the reception for this N-SDU is busy or one or more of the parameters are erroneous (wrong id, wrong type, wrong value). |
| | |
| This service is used to set a new value for the given Parameter (STmin or BS) for the given NSdu-Id. The new value is used for the FCs to be transmitted for each newly started reception until ECU reset. After power on the statically configured default values are restored and used again until a new dynamic change appears.<br>Note: For each call of this API a dedicated call of the "PduR_CanTpChangeParameterConfirmation" callout function is accomplished which delivers the belonging notification result for the requested parameter change (see 5.4.4.4). This confirmation callout function is called from task level during the next task cycle. | |
| | |
| The dynamic setting of the parameters is optional and can be configured in the generation tool (CANTP_ENABLE_CHANGE_PARAM == STD_ON/STD_OFF). The default is STD_OFF.<br>It is not permitted to call this API function several  times consecutively before a belonging "PduR_CanTpChangeParameterConfirmation" callout appeared. The result of the parameter change always has to be awaited in the confirmation function before another change request is performed.<br>Do not call this API function within the "PduR_CanTpChangeParameterConfirmation" callout function. A deadlock situation can appear if the confirmation callout is not finished yet but the next parameter change is already requested.<br>Available since version 1.15.00 | |
| Expected Caller Context | |
| Task context | |

Table 5-14 CanTp_ChangeParamRequest

| | |
|---|---|
| | |
| `uint8 ( PduIdType CanTpRxSduId, TPParameterType Parameter, uint16* Value )` | |
| | |
| CanTpRxSduId | ID of CAN N-SDU for which the new value shall be set |
| Parameter | The parameter (either STmin or BS) which shall be read |
| Value | The value from the selected 'Parameter' |
| | |
| E_OK<br>E_NOT_OK | If the value could be read.<br>If the value could not be read. |
| | |
| This service is used to read the current Value for the given Parameter (STmin or BS) for the given NSdu-Id. | |
| | |
| The reading of the parameters is optional and can be configured in the generation tool (CANTP_ENABLE_CHANGE_PARAM == STD_ON/STD_OFF). The default is STD_OFF. | |
| Expected Caller Context | |
| Task context | |

Table 5-15 CanTp_ReadParamRequest

In the following table services provided by other components, which are used by the CANTP are listed. For details about prototype and functionality refer to the documentation of the providing component.

| | |
|---|---|
| DET | Det_ReportError |
| DEM | Dem_SetEventStatus |
| CANIF | CanIf_Transmit,<br>CanIf_CancelTransmit |
| SCHM | Tp_AsrTpCan_EnterCritical,<br>Tp_AsrTpCan_LeaveCritical |
| ECUM | EcuM_GeneratorCompatibilityError (see [9]) |

Table 5-16 Services used by the CANTP

This chapter describes the callback functions that are implemented by the CANTP and can be invoked by other modules. The prototypes of the callback functions are provided in the header file CanTp_Cbk.h by the CANTP.

| | |
|---|---|
| | |
| `void ( PduIdType CanTpRxPduId,`<br>`const PduInfoType* CanTpRxPduPtr )` | |
| | |
| CanTpRxPduId | ID of CAN N-PDU that has been received. Range: 0..(maximum number of N-PDU IDs received )– 1.<br><br>Basic information about a PDU of any type:<br>`typedef struct`<br>`{`<br>`  uint8         *SduDataPtr;`<br>`  PduLengthType  SduLength;`<br>`} PduInfoType;` |
| CanTpRxPduPtr | Indicator of structure with received N-SDU (payload) and data length. |
| | |
| void | -- |
| | |
| This function is called by the CAN Interface after a successful reception of a Rx CAN N-PDU.<br>This callback service is called by the Can Interface and is implemented by the CANTP. It will be called in 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 N-PDU buffers are not global and are therefore distributed in the corresponding CAN Transport Layer. | |
| | |
| -- | |
| Expected Caller Context | |
| This function may be called if an interrupt occurs (from the CAN receive interrupt). | |

Table 5-17 CanTp_RxIndication

| | |
|---|---|
| void | ( PduIdType CanTpTxPduId ) |
| | |

Please take care of the AUTOSAR Release Version.

In Release 3 another set of PduR API Callout Functions is specified than in the Release 4 version.

Check for the availability by considering the version information in the function description below !

The CANTP does not define any notifications.

At its configurable interfaces the CANTP defines callout functions. The declarations of the callout functions are provided by the BSW module, i.e. the CANTP. It is the integrator's task to provide the corresponding function definitions. The definitions of the callouts can be adjusted to the system's needs. The CANTP callout function declarations are described in the following tables:

| | |
|---|---|
| | |
| `void` | `( PduIdType CanTpRxPduId, NotifResultType Result )` |
| | |
| CanTpRxPduId | ID of CAN N-PDU that has been received.<br>Range: 0..(maximum number of N-PDU IDs which may be received by CAN TP) - 1. |
| Result | Result of the TP reception:<br>NTFRSLT_OK<br>in case TP reception completed successfully.<br>NTFRSLT_E_NOT_OK<br>NTFRSLT_E_TIMEOUT_A<br>NTFRSLT_E_TIMEOUT_Cr<br>NTFRSLT_E_WRONG_SN<br>NTFRSLT_E_UNEXP_PDU<br>NTFRSLT_E_NO_BUFFER<br>in case TP reception did not complete successfully. |
| | |
| void | -- |
| | |
| This function is called by the CAN TP.<br>with Result = NTFRSLT_OK after all CAN TP data has successfully been received (i.e. at the very end of the segmented TP receive cycle or after reception of an unsegemented N-PDU).<br>with Result != NTFRSLT_OK if an error (e.g. timeout) occurred during the TP reception. This enables unlocking of the receive buffer. It is undefined which part of the buffer contains valid data in this case.<br>Note: Since AUTOSAR Release 3.0 (which corresponds to CANTP version 01.07.00) after a reception is completed successfully by calling PduR_CanTpRxIndication, the CANTP calls the DEM with EventId= CANTP_E_COM (see also 3.5.2), EventStatus=DEM_EVENT_STATUS_PREPASSED. If the reception was aborted (e.g. timeout), the CANTP calls the DEM with EventId= CANTP_E_COM (see also 3.5.2), EventStatus=DEM_EVENT_STATUS_PREFAILED. | |
| | |
| Used to enable unlocking of the receive buffer. | |
| Call Context | |

| |
|---|
| This function might be called in interrupt context.<br><br>Care must be taken not to use OS calls in this callout function because the callout function can be called within critical sections. If so, OS calls used in this context will usually lead to an OS error indication (e.g. the OS error hook function is called and the ECU is blocked until a watchdog reset). |

Table 5-19 PduR_CanTpRxIndication

| | |
|---|---|
| | |
| `void` | `( PduIdType CanTpTxPduId,`<br>`NotifResultType Result )` |
| | |
| CanTpTxPduId | ID of CAN N-PDU that has been transmitted.<br>Range: 0..(maximum number of N-PDU IDs which may be transmitted by CAN TP) - 1. |
| Result | Result of the TP transmission:<br>NTFRSLT_OK<br>in case TP transmission completed successfully.<br>NTFRSLT_E_NOT_OK,<br>NTFRSLT_E_TIMEOUT_A,<br>NTFRSLT_E_TIMEOUT_Bs,<br>NTFRSLT_E_INVALID_FS,<br>NTFRSLT_E_WFT_OVRN,<br>NTFRSLT_E_NO_BUFFER<br>in case TP transmission did not complete successfully. |
| | |
| void | -- |
| | |
| This function is called by the CAN Transport Protocol:<br>with Result = NTFRSLT_OK after all CAN TP data has successfully been transmitted, i.e. at the very end of the segmented TP transmission cycle. This is normally done within the CAN Tx Confirmation interrupt.<br>with Result != NTFRSLT_OK if an error (e.g. timeout) occurred during the TP transmission. This enables unlocking of the transmit buffer.<br>Note: Since AUTOSAR Release 3.0 (which corresponds to CantP version 01.07.00) after a transmit is completed successfully by calling PduR_CanTpTxConfirmation, the CANTP calls the DEM with EventId= CANTP_E_COM (see also 3.5.2), EventStatus=DEM_EVENT_STATUS_PREPASSED. If the transmit was aborted (e.g. timeout), the CANTP calls the DEM with EventId= CANTP_E_COM (see also 3.5.2), EventStatus=DEM_EVENT_STATUS_PREFAILED. | |

| |
|---|
| Used to enable unlocking of the transmit buffer. |
| **Call Context** |
| This function might be called in interrupt context (e.g. from CAN transmit interrupt). <br> ⚠ Care must be taken not to use OS calls in this callout function because the callout function can be called within critical sections. If so, OS calls used in this context will usually lead to an OS error indication (e.g. the OS error hook function is called and the ECU is blocked until a watchdog reset). |

Table 5-20 PduR_CanTpTxConfirmation

The following functions are provided for the PduR API defined until AUTOSAR Release 3. Please check chapter 5.4.4 for the Release 3 API.

| | |
|---|---|
| `BufReq_ReturnType` | `( PduIdType CanTpRxPduId,`<br>`PduLengthType TpSduLength,`<br>`PduInfoType **PduInfoPtr )` |
| | |
| CanTpRxPduId | ID of CAN N-PDU that shall be received. Range: 0..(maximum number of N-PDU IDs which may be received by CAN TP) - 1. |
| TpSduLength | This length identifies the overall number of bytes to be received. This parameter will not be changed on subsequent calls of this service requesting a new buffer for the same CanTpRxPduId. The length will be greater than zero. |
| PduInfoPtr | Pointer to pointer to PduInfoStructure containing SDU data pointer and SDU length of a receive buffer. If the return value is not equal to BUFREQ_OK, PduInfoPtr is undefined and shall not be used. |
| | |
| BufReq_ReturnType | BUFREQ_OK Buffer request accomplished successfully<br>BUFREQ_E_BUSY Currently no buffer available<br>BUFREQ_E_OVFL Receiver is not able to receive number of `TpSduLength` bytes; no buffer provided.<br>BUFREQ_E_NOT_OK Buffer request not successful, no buffer provided. |
| | |
| This service is called by the CAN TP for requesting a new buffer (pointer to a PduInfoStructure containing a pointer to a SDU buffer and the buffer length) for the CAN TP to fill in the received data.<br>The length of the buffer does not need to be in the length of the expected SDU. If the returned buffer length is smaller than the expected length, the receiver will be requested via a further call of this service to provide another buffer after the current buffer has been filled up with data. | |
| | |
| By this service the receiver (e.g. DCM) is also informed implicitly about a first frame reception or a single frame reception.<br>After returning a valid buffer, the receiver must not access this buffer unless:<br>- it has been requested to provide a new buffer for the same `CanTpRxPduId`, or<br>- it has been notified (via `PduR_CanTpRxIndication`) about the successful reception (indication), or<br>- it has been notified (via `PduR_CanTpRxIndication`) that the reception was aborted (error indication).<br>When filling the provided buffer, the CAN TP will also set the length information contained in `*PduInfoPtr` to the number of bytes that are valid in this buffer. | |
| Call Context | |
| This function might be called in interrupt context. | |

Table 5-21 PduR_CanTpProvideRxBuffer

| | |
|---|---|
| | |
| `BufReq_ReturnType ( PduIdType CanTpTxPduId,`<br>`PduInfoType **PduInfoPtr,`<br>`uint16 Length )` | |
| | |
| CanTpTxPduId | ID of CAN N-PDU to be transmitted<br>Range: 0..(maximum number of N-PDU IDs which may be transmitted by CAN TP) – 1. |
| PduInfoPtr | Pointer to pointer to PduInfoStructure containing SDU data pointer and SDU length of a transmit buffer. This length must not be smaller than the length given by the ‘Length’ parameter. If the return value of this callback is not equal to BUFREQ_OK then the `PduInfoPtr` is undefined and will not be used by the CANTP.<br><br>Basic information about a PDU of any type:<br>`typedef struct`<br>`{`<br>`  uint8          *SduDataPtr;`<br>`  PduLengthType  SduLength;`<br>`} PduInfoType;` |
| Length | Exact length of the requested transmit buffer, it shall not exceed the number of bytes still to be sent. Length may be zero, which indicates that the provided buffer can be of arbitrary size (larger than zero). |
| | |
| BufReq_ReturnType | BUFREQ_OK Buffer request accomplished successfully,<br>BUFREQ_E_BUSY Currently no buffer of the requested size is available,<br>BUFREQ_E_NOT_OK Buffer request not successful, no buffer provided. |
| | |
| This function is called by the CAN TP for requesting a transmit buffer. The length of the buffer does not need to match the length of the complete N-SDU to be transmitted. It only needs to be as large as required by the caller of that service (`Length`). | |
| | |
| This function might be called in interrupt context.<br>In case this service returns BUFREQ_E_NOT_OK the related transmit request is cancelled and the CAN TP module finishes the request by providing a final confirmation indicating an error.<br>In case this service returns BUFREQ_E_BUSY the related transmit request is delayed and the CAN TP module retries the buffer request. | |
| Call Context | |
| This function could be called if an interrupt occurs (from the CAN receive interrupt). | |

Table 5-22 PduR_CanTpProvideTxBuffer

The following functions are provided for the PduR API defined until AUTOSAR Release 4. Please check chapter 5.4.3 for the Release 3 API.

| | |
|---|---|
| | |
| `BufReq_ReturnType CanTpRxPduId,` | `( PduIdType`<br>`PduLengthType TpSduLength,`<br>`PduLengthType* RxBufferSizePtr )` |
| | |
| CanTpRxPduId | ID of CAN N-PDU that shall be received.<br>Range: 0..(maximum number of N-PDU IDs which may be received by CAN TP) - 1. |
| TpSduLength | Complete length of the TP I-PDU to be received. |
| RxBufferSizePtr | Currently available receive buffer in the upper layer. The available receive buffer might be smaller than TpSduLength depending on the buffering strategy. |
| | |
| BufReq_ReturnType | BUFREQ_OK: The connection is accepted and the RxBufferSizePtr indicates the available receive buffer.<br>BUFREQ_E_BUSY: Currently no buffer of the requested size is available. The RxBufferSizePtr remains unchanged and the connection is rejected.<br>BUFREQ_E_NOT_OK: The connection is rejected and the RxBufferSizePtr remains unchanged. |
| | |
| Called once to initialize the reception of any TP N-PDU. | |
| | |
| By this service the receiver (e.g. DCM) is also informed implicitly about a first frame reception or a single frame reception. | |
| Call Context | |
| This function might be called in interrupt context. | |

Table 5-23 PduR_CanTpStartOfReception

| | |
|---|---|
| | |
| `BufReq_ReturnType ( PduIdType CanTpRxPduId, PduInfoType* PduInfoPtr, PduLengthType* RxBufferSizePtr )` | |
| | |
| CanTpRxPduId | ID of CAN N-PDU that shall be received.<br>Range: 0..(maximum number of N-PDU IDs which may be received by CAN TP) - 1. |
| PduInfoPtr | Pointer to a PduInfoType which indicates the number of bytes to be copied (SduLength) and the location of the source data (SduDataPtr). An SduLength of zero is possible in order to poll the available receive buffer size. In this case no data are to be copied and SduDataPtr might be invalid. |
| RxBufferSizePtr | Remaining receive buffer after completion of this call. |
| | |
| BufReq_ReturnType | BUFREQ_OK: Data has been copied to the receive buffer completely as requested.<br>BUFREQ_E_NOT_OK: Data has not been copied. Request failed. |
| | |
| Called once upon reception of each segment. Within this call the received data is copied to the receive buffer. | |
| | |
| The function must only be called if the connection has been accepted by an initial call to PduR_<bus>TpStartOfReception.<br>The API may only be called with an SduLength greater zero if the RxBufferSizePtr returned by the previous API call indicates sufficient receive buffer (SduLength <= RxBufferSizePtr). | |
| Call Context | |
| This function might be called in interrupt context. | |

Table 5-24 PduR_CanTpCopyRxData

| | |
|---|---|
| | |
| `BufReq_ReturnType ( PduIdType CanTpTxPduId, PduInfoType* PduInfoPtr, RetryInfoType* RetryInfoPtr, PduLengthType* TxDataCntPtr )` | |
| | |
| CanTpTxPduId | ID of CAN N-PDU that shall be transmitted.<br>Range: 0 .. (maximum number of N-PDU IDs which may be transmitted by CAN TP) - 1. |
| PduInfoPtr | Pointer to a PduInfoType which indicates the number of bytes to be copied (SduLength) and the location where the data have to be copied to (SduDataPtr).<br>An SduLength of zero is possible in order to poll the available transmit data count. In this case no data are to be copied and SduDataPtr might be invalid. |
| RetryInfoPtr | The CANTP does not support the retry feature. So, either a NULL_PTR can be provided or alternatively TpDataState can indicate TP_NORETRY.<br>Both methods indicate that the copied transmit data can be removed from the buffer after it has been copied. |
| TxDataCntPtr | Remaining Tx data after completion of this call. |
| | |
| BufReq_ReturnType | BUFREQ_OK: Data has been copied to the transmit buffer completely as requested.<br>BUFREQ_E_BUSY: Request could not be fulfilled as the required amount of Tx data is not available. The CANTP might retry later on. No data has been copied.<br>BUFREQ_E_NOT_OK: Data has not been copied. Request failed. |
| | |
| Called by the TP layer to retrieve the payload of a Tx TP I-PDU segment. The data are copied within this call. | |
| | |
| The already available Tx data can be queried by calling PduR_CanTpCopyTxData with an SduLength of zero after the transmission request was received. | |
| Call Context | |
| This function might be called in interrupt context. | |

Table 5-25 PduR_CanTpCopyTxData

| | |
|---|---|
| `void (PduIdType CanTpRxSduId, NotifResultType Result )` | |
| | |
| CanTpRxSduId | ID of CAN N-SDU that was requested to be changed.<br>Range: 0 .. (maximum number of N-SDU IDs which may be received by CAN TP) - 1. |
| Result | Can be on of the following notifications:<br>- NTFRSLT_PARAMETER_OK: The parameter change request has been successfully executed.<br>- NTFRSLT_E_PARAMETER_NOT_OK: The request for the change of the parameter did not complete successfully.<br>- NTFRSLT_E_RX_ON: The parameter change request not executed successfully due to an ongoing reception.<br>- NTFRSLT_E_VALUE_NOT_OK: The parameter change request not executed successfully due to a wrong value. |
| | |
| none | |
| | |
| Called once by the CANTP to indicate the result of a CanTp_ChangeParameterRequest call. | |
| | |
| Available since version 1.15.00<br>In case of multiple change parameter requests appearing without awaiting the confirmation callout function the confirmation callout function is called synchronously from within the CanTp_ChangeParameterRequest API function. | |
| Call Context | |
| This function is called from the CANTP main function in task context. | |

Table 5-26 PduR_CanTpChangeParameterConfirmation

| | |
|---|---|
| `void (uint16 CanTpModuleId, uint8 CanTpInstanceId )` | |
| | |
| CanTpModuleId | Contains the CANTP_MODULE_ID (23 hex) as defined by AUTOSAR. |
| CanTpInstanceId | For the CanTp only one instance is available, so this value is always zero. |
| | |
| none | |
| | |
| Called once by the CANTP during the initialization phase to indicate one of the following possible errors:<br>- Abort initialization as generator is not compatible,<br>- Abort initialization as current configuration is not compatible with the pre-compile configuration<br>- Abort initialization as current configuration is not compatible with the link-time configuration | |
| | |
| None | |
| Call Context | |
| This function is called in the CanTp_Init function during power on. | |

Table 5-27 EcuM_GeneratorCompatibilityError

In the CANTP the attributes can be configured with the following methods:

Configuration in Database, for a detailed description see 6.1

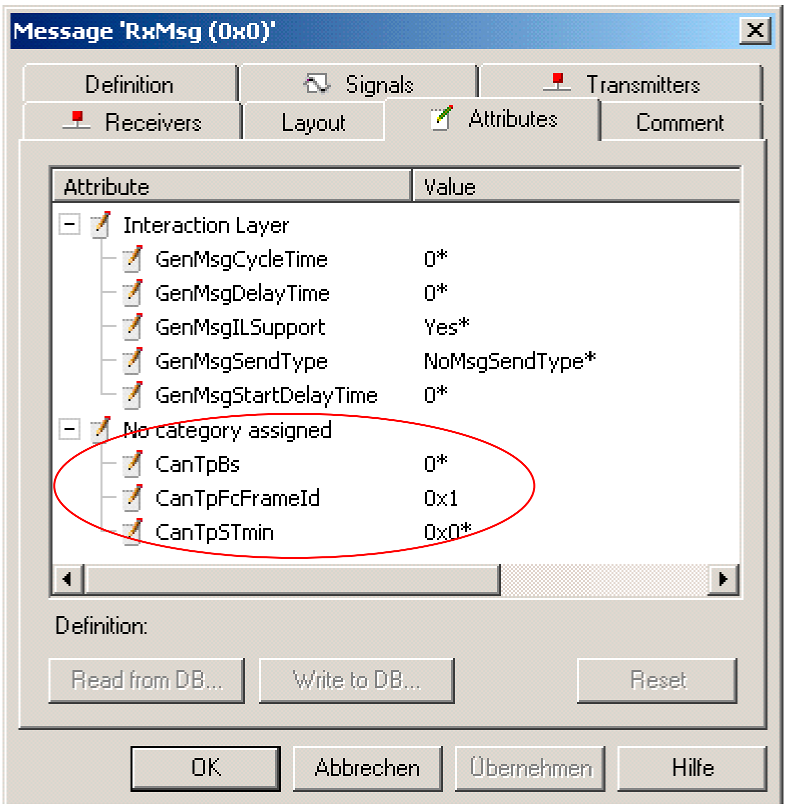
Configuration in GENy for a detailed description see 6.2

The following database attributes are utilized by the generation tool for the CANTP:

| | | | the is written in bold | |
|---|---|---|---|---|
| CanTpFcFrameId | Message | Integer | 0x 0000 0000 … 0x1FFF FFFF | Required from large COM PDUs only. For all database messages with this attribute the generation tool automatically creates a Rx-Sdu (for Tx messages) or a Tx-Sdu (for Rx messages). The message Identifier itself is used within these SDUs as CANTP_TXFC_NPDU_ID or CANTP_RXFC_NPDU_ID as corresponding Flow-Control message identifier. Only evaluated for messages with DLC > 8. |
| CanTpBs | Message | Integer | … 0xFF | Required from large COM PDUs only. Number of consecutive frames (blocksize) to be received without intermittent flow control frames. Only evaluated for messages with DLC > 8. |
| CanTpSTmin | Message | Integer | …0x7F 0xF1 …0xF9 | Required from large COM PDUs only. Minimum time distance between two messages to be received by the CANTP. Only evaluated for messages with DLC > 8. |

Table 6-1 Data base attributes

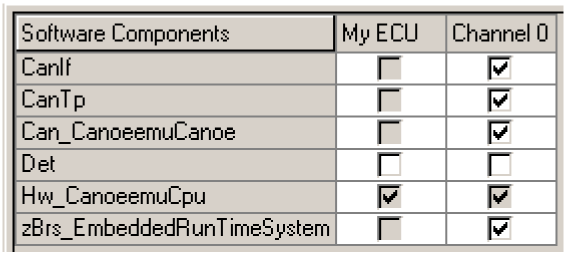
Example for an Rx message:



| Attribute | Value |
|---|---|
| Interaction Layer | |
| GenMsgCycleTime | 0* |
| GenMsgDelayTime | 0* |
| GenMsgILSupport | Yes* |
| GenMsgSendType | NoMsgSendType* |
| GenMsgStartDelayTime | 0* |
| No category assigned | |
| CanTpBs | 0* |
| CanTpFcFrameId | 0x1 |
| CanTpSTmin | 0x0* |

This section describes how to configure CANTP with the help of the configuration tool GENy. Some general configuration aspects like the configuration variant, DEM and or DET usage or customer specific optimization requirements are chosen in first step prior to the configuration of the attributes of each single Rx/Tx- NSdu.

To use the CANTP layer the component must be enabled in the component selection dialog of the system configuration as shown in the following screenshot:



| Software Components | My ECU | Channel 0 |
|---|---|---|
| CanIf | ☐ | ☑ |
| CanTp | ☐ | ☑ |
| Can_CanoeemuCanoe | ☐ | ☑ |
| Det | ☐ | ☐ |
| Hw_CanoeemuCpu | ☑ | ☑ |
| zBrs_EmbeddedRunTimeSystem | ☐ | ☑ |

Typically, beside the CAN Driver itself for a physical bus access, the CANTP is surrounded by the CanIf on the lower layer and the PduR on the upper layer side.

| | | The default value is written in bold | |
|---|---|---|---|
| Configuration Variant | enumeration | customer specific pre-configuration | Enter the configuration method to be used - either precompile or postbuild. A link-time configuration variant is not specified for CANTP (see [1]). |
| CanTp Main Function Period | uint8 | | This Task Cycle determines the time period for the CanTp_MainFunction scheduling (as milliseconds). |
| CheckCanTp VersionInfoApi | boolean | OFF | Enable the CanTp_GetVersionInfo() API function. |
| CanTpDevError Detect | boolean | ON | Enable the detection and reporting of development errors via DET. (DET has to be available in the system). |
| Production Error Detection | boolean | ON | Enable the detection and reporting of production errors via DEM. (DEM has to be available in the system). |
| User Config File | string | | Use a user configuration file. The content of this file can be used to define specific precompile switches not available otherwise. |
| Support frame padding | boolean | OFF | Enable padding activation. |
| Support RxIndication | boolean | OFF | Support the function CanTp_RxIndication (using a pre-compile macro) |
| Support TxConfirmation | boolean | OFF | Support the function CanTp_TxConfirmation (using a pre-compile macro). |
| Support-StandardAddressing, ExtendedAddressing Mixed11Addressing Format | boolean | OFF | Either the usage of STANDARD and/or EXTENDED and/or MIXED-11 addressing can be configured. If only one addressing type is required for a project then resources can be saved by deselecting the other ones. |
| Use InitCfg pointer | boolean | ON | Enable the CanTp_Init API function with configuration pointer parameter usage (see 7.1). This option is usually pre-configured. It is set to enable but invisible by default.<br>Optionally (non AUTOSAR) available feature for the parameterization of the CanTp_Init function call. This is only relevant for AUTOSAR 3. Since AUTOSAR 4 this is covered by AUTOSAR itself.<br><br>Please note that the Multiple Configuration feature requires the usage of this configuration pointer. Via the configuration pointer the actual identity is, typically during power on, selected. |
| Module Start Address | integer | | Enter the configuration structure start address. |
| CanTpMaxNum<br><br>RxSdus | integer | 1 | Defines the maximum number of different N-Sdu-Ids that can be received simultaneously via the CanTp by this ECU. |

| | | The default value is written in bold | |
|---|---|---|---|
| CanTpMaxNum TxSdus | integer | 1 | Defines the maximum number of different N-Sdu-Ids that can be transmitted simultaneously via the CanTp by this ECU. |
| CanTpMaxNum RxChannels_PbLimit ' | integer | 3 | Defines the maximum for parameter 'CanTpMaxNumRxChannels' that can be set during post build configuration. This setting only has an effect if "CanTpDynamicChannelAssignment" is enabled . <br><br> Please note that this value only serves as a reference value and cannot be modified. If the actual number of dynamic channels does not properly reflect the number of statically configured Sdus during a later postbuild phase then the user is warned against this conflict: “the 'CanTpMaxNumRxChannels' attribute is not post-build capable”. |
| CanTpMaxNum TxChannels_PbLimit' | integer | 3 | Defines the maximum for parameter 'CanTpMaxNumTxChannels' that can be set during post build configuration. This setting only has an effect if "CanTpDynamicChannelAssignment" is enabled. <br><br> Please note that this value only serves as a reference value and cannot be modified. If the actual number of dynamic channels does not properly reflect the number of statically configured Sdus during a later postbuild phase then the user is warned against this conflict: “the 'CanTpMaxNumTxChannels' attribute is not post-build capable”. |
| CanTpTc | boolean | ON | Preprocessor switch for enabling a Transmit Cancellation. By enabling this switch an ongoing transmission request can be aborted using the CanTp_CancelTransmitRequest API. |
| CanTpTc NotifyAlways | booelean | ON | Switches upper layer notification of transmit cancellation ON or OFF. <br><br> This additional non AUTOSAR switch is used for compatibility reasons. <br><br> In case there are upper layer modules (e.g. DCM) which require a final indication or confirmation, this switch may be used to hand a final negative notification (NTFRSLT_E_CANCELATION_NOT_OK) to the upper layer which has requested the cancellation instead of just stopping without further notification. |
| CanTpRc | boolean | ON | Preprocessor switch for enabling a Receive Cancellation. By enabling this switch an ongoing reception can be aborted using the CanTp_CancelReceiveRequest API. |
| CanTpRc | booelean | ON | Switches upper layer notification of receive |

| | | The default value is written in bold | |
|---|---|---|---|
| NotifyAlways | | | cancellation ON or OFF.<br>This additional non AUTOSAR switch is used for compatibility reasons.<br>In case there are upper layer modules (e.g. DCM) which require a final indication, this switch may be used to hand a final negative notification (NTFRSLT_E_CANCELATION_NOT_OK) to the upper layer which has requested the cancellation instead of just stopping without further notification. |
| Support Frame Padding pattern | boolean | ON | Enable optional padding byte definition in case of padding is activated. |
| Frame Padding pattern | uint8 | 0xAA | Define the padding pattern if “Support Frame Padding” is enabled. |
| Enable Split MainFunction | boolean | ON | When selecting this optional feature then the CanTp_MainFunction is split into two separate Functions for the Rx and the Tx path, CanTp_MainFunctionRx and CanTp_MainFunctionTx (see 5.1.10, 5.1.12). These separate API functions are intended to be called instead of the CanTp_MainFunction to optimize the processing sequence of incoming requests (e.g.: receive a diagnostic request, process the request, transmit the response). |
| Enable Rx/Tx NSduId Remapping | boolean | ON | When enabling this optional feature an additional entry for each Tx-NSdu is provided to the user wherein a user specific NSdu-Id can be entered. On the Rx side the NSdu-Ids are determined by the PDUR.<br>If not selected then internally generated NSdu-Id number are used (and handed to the upper layers).<br>Via preconfiguration files the optional and OEM specific Wrapper functions CanTpWrapperTransmit and CanTpWrapper_CancelTransmitRequest can be enabled. These wrapper functions are parameterized with the remapped N-SDU-Ids instead of the internally generated IDs. Apart from that the API of the wrapper functions is equivalent to the AUTOSAR CanTp_Transmit respectively CanTp_CancelTransmitRequest API function (see 5.1.6, 5.1.7). |
| Enable Extended API BS/STmin | boolean | ON | When enabling this optional feature then the Flow Control frame (FC) content for blocksize (BS) and minimum separation time (STmin), which is responded to the sender upon reception of a First or Consecutive frame, can be modified during runtime. Therefore additional API functions are provided to set the FC content at any time of the execution. |
| CanTpChange | booelean | ON | When switched on, the functions |

| | | The default value is written in bold | |
|---|---|---|---|
| ParameterReqApi | | | "CanTp_ChangeParameterRequest" and "CanTp_ReadParameterRequest" are provided which can be used to change and read the values of the reception parameters CanTpBs and CanTpSTmin.<br><br>If switched on, the PduR module must provide the function "PduR_CanTpChangeParameterConfirmation". |
| Select the API towards the PduR to be used (CanTpAr40CompliantPduRApi) | boolean | ON, | If the Vector PduR is present then the PduR provided API is used.<br><br>If no Vector PduR is present then this value must be preconfigured or enabled within CanTp.<br><br>Note: This is only relevant for AUTOSAR 3. When using an AUTOSAR 4 environment then the PduR API is always AUTOSAR 4 compliant. |
| Support Full Buffer handling (on Rx and/or Tx- side) | boolean | OFF | When the full buffer handling is disabled then only the complete requested buffer size (no partial buffer handling) is accepted from the CANTP, otherwise the connection is terminated after the “Provide(Rx/Tx)Buffer” request. Within AUTOSAR the buffer provision is handled highly dynamical. Repeated buffer provision calls to the application offer the opportunity to the application to deliver different sizes of application buffers that are temporarily available. |
| Single connection optimization | boolean | ON | When selecting this optimization then only two Rx- and one Tx-NSdu  can be configured. The number of totally configurable connections is “1” since the second Rx-NSdu is only implemented as an additional so called “functional reception path” for diagnostic applications. Selecting this checkbox still keeps the relevant information applicable for postbuild processing in opposite to next topic “Single connection optimization precompile” (see below). |
| Single connection optimization precompile | boolean | ON | When selecting this optimization then only two Rx- and one Tx-NSdu  can be configured. The number of totally configurable connections is “1” since the second Rx-NSdu is only implemented as an additional so called “functional reception path” for diagnostic applications. In addition the connection attributes are generated as precompile defines. Please note that this causes the unavailability of post-build functionality. The NSdu-attributes are contained in the component code and can not be modified without a re-compilation. Please not in addition that this causes also that Multiple Configuration functionality (see 6.2.3) is absent. |
| Support dynamic channel assignment | boolean | ON | For a given number of statically configured Rx/Tx-NSdus a maximum number of parallel available CANTP channel resources can be specified. This |

| | | The default value is written in bold | |
|---|---|---|---|
| | | | maximum number defines how many connections can be active simultaneously. So the maximum number of simultaneously active Rx and Tx channels will be restricted when this feature is active. When this feature is disabled, each Rx/Tx-NSdu can be active at the same time. |
| CanTpMaxNum[Rx/Tx]Channels | uint16 | 1/1 | This is the maximum number of simultaneously available channels. It's only relevant if the "Dynamic Channel Assignment" optimization is selected. Please note that it is not allowed to specify a maximum channel number which is higher then the total number of available NSdus. |

Table 6-2 General parameters

| | | The default value is written in bold | |
|---|---|---|---|
| CanTpRxNPdu | PduIdType | -- | Select the CAN message to be used for this N-PDU reception. |
| CanTpTxFCNPdu | PduIdType | -- | Select the CAN message to be used in case of FC responses for this N-PDU. |
| CanTpRxPadding Activation | boolean | OFF | Enable padding. If so, all PDUs must be of length 8. |
| CanTpRxTaType | enumeration | CANTP_FUNCTIONAL | Select the communication type, either physical for 1:1 connections or functional for 1:n connections. |
| Addressing Format | enumeration | CANTP_EXTENDED, CANTP_MIXED11 | Select the addressing format. If extended addressing is used, matching Target/Source Addresses (see below) must be defined too. If Mixed-11 addressing is used, matching Address Extension information (see below) must be defined too. |
| CanTpBs | uint8 | | Enter the maximum Block size. Please note that this BS value can be adapted at runtime due to buffer availability. |
| CanTpSTmin | uint8 | | Enter the minimum time between two consecutive frames (milliseconds). |
| CanTpRxWftMax | uint16 | | Enter the maximum number of Wait Frames that are consecutively received before an error notification is raised and the communication is |

| | | The default value is written in bold | |
|---|---|---|---|
| | | | stopped for this N-SDU. |
| CanTpRxDl | uint16 | | The minimum Data Length Code is determined automatically. If padding is active, this must be 8. If padding is not active, a minimum size of 1 is assumed. All messages with a length less than this Data Length Code are ignored. |
| CanTpNar | uint16 | | Enter the CAN confirmation timeout (milliseconds). |
| CanTpNbr | uint16 | | Enter the Flow Control timeout (milliseconds). |
| CanTpNcr | uint16 | | Enter the Consecutive Frame timeout (milliseconds). |
| CanTpNTa / CanTpNAe | uint8 | | Enter the Target Address in case of extended addressing format. Enter the Address Extension in case of mixed-11 addressing format. Please note that the parameter CanTpNAe does not exist in the BSWMD file, instead of CanTpNTa is used to store the address extension. |
| CanTpNSa | uint8 | | Enter the Source Address to be used for FC-responses for this N-PDU if extended addressing is selected. |
| Rx Upper Layer NSdu Id | PduIdType | -- | This is the NSDU-ID which will be used for the upper layer communication in case of an enabled Rx/Tx NSduId remapping. |

Table 6-3 Rx parameters

| | | The default value is written in bold | |
|---|---|---|---|
| CanTpTxNPdu | PduIdType | -- | Select the CAN message to be used for this N-PDU transmission. |
| CanTpRxFCNPdu | PduIdType | -- | Select the CAN message to be used in case of FC receptions for this N-PDU. |
| CanTpTxPadding Activation | boolean | OFF | Enable padding. If so, all PDUs must be of length 8 (see 7.1.8 Limitations). |
| CanTpTxTaType | enumeration | CANTP_FUNCTIONAL | Select the communication type, either physical for 1:1 connections or functional for 1:n connections. |
| Addressing Format | enumeration | CANTP_EXTENDED, | Select the addressing format. If extended addressing is used matching Target/Source Addresses (see below) must be defined too. |

|  |  | The default value is written in bold |  |
|---|---|---|---|
|  |  | CANTP_MIXED11 | If Mixed-11 addressing is used then the belonging Address Extension information (see below) must be defined too. |
| CanTpNas | uint16 |  | Enter the CAN confirmation timeout (milliseconds). |
| CanTpNbs | uint16 |  | Enter the Flow Control timeout (milliseconds). |
| CanTpNcs | uint16 |  | Enter the Consecutive Frame timeout (milliseconds). |
| CanTpTxDl | uint16 |  | The minimum Data Length Code is determined automatically. When padding is active this must be 8. When padding is not active, a minimum size of 1 is assumed. All messages with a length less than this Data Length Code are ignored.<br>See also ‘7.1.8 Limitations’ ! |
| CanTpTc | uint8 |  | Select the onnection specific transmit cancellation. |
| CanTpNTa /<br>CanTpNAe | uint8 |  | Enter the Target Address in case of extended addressing format.<br>Enter the Address Extension in case of mixed-11 addressing format. Please note that the parameter CanTpNAe does not exist in the BSWMD file, instead of  CanTpNTa is used to store the address extension. |
| CanTpNSa | uint8 |  | Enter the Source Address to be used for FC-receptions for this N-PDU if extended addressing format is selected. |
| Tx Upper Layer NSdu Id | PduIdType | -- | Enter the NSDU-ID which shall be used for the upper layer communication. |

Table 6-4 Tx parameters

Multiple ECUs are control units which are assembled several times within the CAN network using the identical software (e.g.: front seat left and right hand side). In this case the application has to decide at run-time which ECU is actually installed and has to set-up the communication parameters dynamically. This is typically done during system startup in the CanTp_Init function.

The CanTp_Init is called for this purpose with an identity specific configuration  pointer which selects the actual identity (see 5.1.2).

Example for an ECU identity :

Check the generated file CanTp_PBcfg.h for the belonging definitions:

```
#define                                    vCanTp_Cfg
#define                                    vCanTp_Cfg_1
  ...
```

and then call the init function with the appropriate configuration pointer of the identity you are looking for:

```
CanTp_Init(&CanTp_Config_Identity_X);
```

The CANTP needs all timings to be normalized on call cycles. Therefore all timings will be rounded up to the next integer multiple of call cycles.

Since timers are set within interrupt context and decremented within task-context, some jitter may occur. However, this jitter does not exceed one call cycle.

As for CANTP, the ‘Timings’ are generally calculated with a jitter plus one call cycle, this means that the timing value is the first possible time after an event (e.g. timeout) can occur.

CANTP uses the following algorithm for calculation:

timeValue = (requestedValue + (CallCycle-1)) / CallCycle + 1

As opposed to CANTP289 “The number of connection channels is not directly configurable. It will be determined by the configuration tools during the configuration process, by analyzing the N-SDU/Channel routing table.” the number of dynamically supported connection channels is configurable within the generation tool.

As opposed to CANTP082: “If the function PduR_CanTpStartOfReception() returns BUFREQ_E_BUSY to the CanTp module, the CanTp module shall suspend the N-SDU reception by sending the next Flow Control N-PDU with status WAIT (i.e. FC(WT))“ the Vector CanTp tries further buffer requests using the PduR_CopyRxData API before sending Flow Control frames with status “WAIT”.

(see also the PduR Spec. for AR4 for the expected behaviour of CanTpStartOfReception() when returning BUFREQ_E_BUSY (“BUFREQ_E_BUSY: Currently no buffer of the requested size is available. RxBufferSizePtr remains unchanged. Connection has been rejected”)

In extension to the CANTP SWS [1] the Vector CANTP supports a burst transmission. If the the STmin time is set equal to zero then all CFs are sent directly in the context of the confirmation function.

In opposition to the CANTP SWS [1] the Vector CANTP supports only full duplex communication.

In extension to the CANTP SWS [1] the CanTp_Init() API is also available for the usage with several sets of configurations. It is configurable, via the pre-compile switch CANTP_HAVE_INIT_CFG_PTR, to be called with a pointer to one specific configuration set.

```
#if (CANTP_HAVE_INIT_CFG_PTR == STD_ON)
 the following API is used:
      void CanTp_Init(const void* const pCfgPtr)
  #endif

#if (CANTP_HAVE_INIT_CFG_PTR == STD_OFF)
```

```
 the following API is used:
        void CanTp_Init(void)
   #endif
```

Please note that since AUTOSAR4 the CanTp_Init API function is defined as follows:

void CanTp_Init( const CanTp_ConfigType* CfgPtr )

In extension to the CANTP SWS [1] the Vector CANTP supports, in case of padding support (CANTP_PADDING_ACTIVATION = STD_ON), an additional padding pattern (CANTP_PADDING_PATTERN) that may be used to fill up unused frame bytes. If unused, the frame is filled up with uninitialized data.

In extension to the CANTP SWS [1] the Vector CANTP supports, in case of a splitted main function is configured (CANTP_RXTX_MAINFUNCTION_API = STD_ON), two additional API functions CanTp_MainFunctionRx and CanTp_MainFunctionTx. These additional functions can be used instead of the original AUTOSAR CanTp_MainFunction API (which is still supported) to optimize the calling sequence of incoming requests and their responses.

In extension to the CANTP SWS [1] the Vector CANTP supports, in case of dynamic Flow Control content setting is configured (CANTP_ENABLE_EXT_API_BS/ CANTP_ENABLE_EXT_API_STMIN = STD_ON), two additional API functions which allow the modification of the blocksize (BS) as well as the minimum separation time (STmin) byte contained in the FCs which are responded to a transmitter during runtime.

In opposition to the CANTP SWS [1] the Vector CANTP supports the deactivation of DEM events which are handed to the DEM on each successful completed reception or transmission as described in CanTp228:

After a task is completed successfully by calling PduR_TxConfirmation() or PduR_RxIndication(), the CanTp module shall call the DEM with EventId= CANTP_E_COM (see also 3.5.2) and  EventStatus = DEM_EVENT_STATUS_PREPASSED.”

If the pre-configurable attribute  “AsrCanTp_SuppressPrePassIndication” is configured to be true, then the DEM notification is turned off for all PRE_PASSED events. The concerning preprocessor switch is named “CANTP_SUPPRESS_PREPASS_INDICATION = STD_ON/STD_OFF). This is done to reduce the DEM buffer resources.

If the pre-configurable attribute  “AsrCanTp_FFProvide2RxBufferCalls” is configured to be true, then a special additional second First Frame buffer provision call is performed. This second call is used as a notification to the PduR so the First Frame of a multi frame message can be routed immediately before any provided buffers are completely filled up.

The concerning preprocessor switch is named “CANTP_FF_PROVIDE_2_RXBUFFER_CALLS = STD_ON/STD_OFF). This is done to speed up any multi frame receptions and is used typically in gateway applications.

When using the pre-compile configuration variant then all Rx- and Tx- Sdu attributes are checked for analogousness. For each attribute (e.g. addressing mode, timeout value, …) being identical for all Sdus on the Rx- or the Tx- side one single preprocessor definition is generated instead of several ROM table entries for each Sdu separately. So, assuming many attributes have the same values, a considerable amount of ROM space can be economized. This is appropriate especially for configurations where a lot of Sdus are available as in gateway applications.

An additional - non AUTOSAR – callout function is supported to give the DCM the possibility to check for parallel access from another tester. This additional callout is provided if the pre-compile switch  ‘CANTP_DCM_REQUEST_DETECT’ is set to ‘STD_ON’. This can be accomplished via a preconfiguration file entry for the ‘AsrCanTp_EnableDcmRequestDetect’ generation tool attribute.

If the callout function is enabled in this manner, then each Single Frame or First Frame indicated to the CanTp via the CanIf API function ‘CanTp_RxIndication’ is also indicated to the DCM if the addressing type is Mixed-11.

The callout function prototype is defined as follows:

```
void Dcm_OnRequestDetection(PduIdType CanTpRxPduId, uint8 N_AE);
```

Due to performance reasons the CanTp_Transmit API calls synchronously the Tx buffer provision and, if successful, subsequently the CanIf_Transmit API to accomplish a transmission on the CAN bus without any delay.

Some third party components do not handle this synchronous calling sequence, thus an asynchronous calling is supported too. By setting the pre-compile switch ‘CANTP_ASYNC_TRANSMIT’ (generation tool attribute ‘AsrCanTp_EnableSynchronousTransmit’) to ‘STD_ON’ this behaviour can be enabled. If so, the CanTp_Transmit just takes the transmission request and then returns without any further processing. The Tx buffer provision and a subsequent call to the CanIf_Transmit API is accomplished from the CanTp task level during the next task cycle.

The following special, non AUTOSAR compliant, handling is provided:

The CanTp will ignore each received FC frame if the STmin byte contains a reserved value (0x80 .. 0xF0, 0xFA .. 0xFF).

When using the standard implementation compliant to ISO 157675-2, then the CanTp replaces the reserved STmin value and runs the CanTp connection with the slowest value for the timing (STmin = 127msec).

When using the non compliant implementation, then the CanTp ignores each FC frame containing a reserved STmin value. To enable this behaviour the pre-compile switch 'CANTP_IGNORE_FC_WITH_RES_STMIN' (generation tool attribute 'AsrCanTp_IgnoreFcWithReservedStMin') must be set to 'STD_ON'. The FC is skipped without any notification to the application.

The CanTp will ignore each received FC frame if the Flow Status nibble contains either FC.OVFLW (0x02) or a reserved value (0x03 .. 0x0F).

When using the standard implementation compliant to ISO 157675-2, then the CanTp stops the transmission of a message when a FC.OVFLW (0x02) or an invalid Flow Status (0x03 .. 0x0F) is received with either the notification "NTFRSLT_E_NO_BUFFER" in case of FS = 0x02 (OVFLW) or the notification "NTFRSLT_E_INVALID_FS" in case of FS = 0x03 .. 0x0F.

When using the non compliant implementation, then the CanTp ignores each FC frame containing a Flow Status greater than 0x02. To enable this behaviour the pre-compile switch 'CANTP_IGNORE_FC_OVFL_INVALID_FS' (generation tool attribute 'AsrCanTp_IgnoreFcOvflAndInvalidFS') must be set to 'STD_ON'. The FC is skipped without any notification to the application.

The CanTp will ignore each received CF if the SN is not equal to the expected one.

The current implementation behaves compliant to ISO 157675-2 and thus stops a running connection with the notification "NTFRSLT_E_WRONG_SN".

When using the non compliant implementation, then the CanTp ignores each CF containing a wrong SN. To enable this behaviour the pre-compile switch 'CANTP_IGNORE_CF_WITH_WRONG_SN' (generation tool attribute 'AsrCanTp_IgnoreCfWithWrongSN') must be set to 'STD_ON'. The CF is skipped without any notification to the application.

The reporting of the production error CANTP_E_COM (see also 3.5.2) to the DEM is configurable in the way that the reporting can be switched to the DET. In this case the DET must be enabled or no reporting will be performed.

If the pre-compile switch ‘CANTP_E_COMM_REPORT_DEM’ (AUTOSAR parameter 'CanTpECommReportDem') is set to ‘STD_ON’ then the normal reporting to the DEM takes place.

If the pre-compile switch ‘CANTP_E_COMM_REPORT_DEM’ is set to ‘STD_OFF’ then the reporting is switched to the DET (with the error code 0xB0).

The reporting of the production error CANTP_E_OPER_NOT_SUPPORTED to the DEM is configurable in the way that the reporting can be switched to the DET. In this case the DET must be enabled or no reporting will be performed.

If the pre-compile switch ‘CANTP_OPER_NOT_SUPPORTED_REPORT_DEM’ (AUTOSAR parameter ' CanTpOperNotSupportedReportDem ') is set to ‘STD_ON’ then the normal reporting to the DEM takes place.

If the pre-compile switch ‘‘CANTP_OPER_NOT_SUPPORTED_REPORT_DEM’ is set to ‘STD_OFF’ then the reporting is switched to the DET (with the error code 0xA0).

Memory optimization by the linker via data scattering:

Some linkers allow filling the “holes” caused by padding with some “foreign” data. Since the generation tool cannot be aware of “foreign” data, such optimizations must be disabled.

The AUTOSAR feature of variable block sizes to be requested within Flow Control frames during a pending reception is not compatible with ISO 15765-2 [4].

Usage of mixed addressing within the same connection is not allowed.

The addressing format – either CANTP_STANDARD or CANTP_EXTENDED or CANTP_MIXED11 – must be identical for each pair of Rx/Tx N-SDUs assigned to the same connection

Timing calculation is runtime optimized

Flow Control timeout on receive side:
ISO 15765-2 specification defines the following performance requirement: (N_Br+N_Ar) < 0.9*N_Bs timeout. Here Vector uses 80% of N_Bs timeout instead of 90% due to performance reasons.

The CANTP is able to manage several connections simultaneously.

Concurrent connections can be configured for the CANTP.

All connection-specific information (Channel number, Timing parameter, …) is configured inside the CANTP module.

A connection channel represents an internal path for transmission or reception of the N-SDU. It uses its own resources such as internal buffer, timer or state machine. The connection channels are only for CANTP internal use and not accessible externally.

Each N-SDU is (statically) linked to one connection channel.

Each connection channel is independent from the other connection channels.

The CANTP routes the N-SDU through the correctly configured connection channel. This mechanism does not allow to receive or transmit an N-SDU with the same identifier in parallel, because each N-SDU is linked to only one connection channel.

The compile-time assignment of connection channels to N-SDUs can not be altered via post-build configuration.

The AUTOSAR requirement claiming that a connection channel may be attached to one or more N-SDUs is considered only if the Dynamic Channel Assignment optimization is available (coming up with version 1.09.00) and configured to be active.

If so, the following points are considered:

If a connection channel is assigned to multiple N-SDUs, resources are shared between different N-SDUs and the CANTP will only reject transmission or abort receiving as long as no free connection channels are available.

The number of connection channels is configurable.

| | |
|---|---|
| Buffer | A buffer in a memory area normally in the RAM. It is an area, that the application has reserved for data storage. |
| Callback function | This is a function provided by an application. E.g. the CAN Driver calls a callback function to allow the application to control some action, to make decisions at runtime and to influence the work of the driver. |
| CAN Driver | The CAN Driver encapsulates a specific CAN controller handling. It consists of algorithms for hardware initialization, CAN message transmission and reception. The application interface supports both event and polling notification and WR/RD access to the message buffers. |
| CAN message | Frame which is composed of the start-of-frame, arbitration, control, data , CRC, acknowledge and end-of-frame bit fields. |
| CANbedded | … is the trademark of the Vector communication stack. |
| Channel | A channel defines the assignment (1:1) between a physical communication interface and a physical layer on which different modules are connected to (either CAN or LIN). 1 channel consists of 1..X network(s). |
| Communication stack | The communication stack consists of the communication configuration and the communication kernel, a number of adaptive software components that cover the basic communication requirements in distributed automotive applications. |
| Component | CAN Driver, Network Management ... are software COMPONENTS in contrast to the expression module, which describes an ECU. |
| Confirmation | A service primitive defined in the ISO/OSI Reference Model (ISO 7498). With the service primitive 'confirmation' a service provider informs a service user about the result of a preceding service request of the service user. Notification by the CAN Driver on asynchronous successful transmission of a CAN message. |
| Critical section | A critical section is a sequence of instructions where mutual exclusion must be ensured. Such a section is called 'critical' because shared data is modified within it. |
| Data consistency | Data consistency means that the content of a given application message correlates unambiguously to the operation performed onto the message by the application. This means that no unforeseen sequence of operations may alter the content of a message hence rendering a message inconsistent with respect to its allowed and expected value. |
| Data link layer | The communication layer which provides services for the transfer of data link messages. The data link layer consists of the communication hardware and the communication driver software. |
| Deadlock | A state in which tasks block one another so that further processing of the tasks concerned is no longer possible. A deadlock between two tasks |

| | |
|---|---|
| | occurs, e.g. if both tasks wait for the reception of a message which is to be sent by the other task before sending its own message. |
| Electronic Control Unit | Also known as ECU. Small embedded computer system consisting of at least one CPU and corresponding periphery which is placed in one housing. |
| Event | An exclusive signal which is assigned to a certain extended task. An event can be used to send a binary information to an extended task. The meaning of events is defined by the application. Any task can set an event for an extended task. The event can only be cleared by the task which is assigned to the event. |
| Gateway | A gateway is designed to enable communication between different bus systems, e.g. from CAN to LIN. |
| GENy | Generation tool for CANbedded and MICROSAR components |
| Indication | A service primitive defined in the ISO/OSI Reference Model (ISO 7498). With the service primitive 'indication' a service provider informs a service user about the occurrence of either an internal event or a service request issued by another service user. Notification of application in case of events in the Vector software components, e.g. an asynchronous reception of a CAN message. |
| Interrupt | Processor-specific event which can interrupt the execution of a current program section. |
| Interrupt level | Processing level provided for time-critical activities. To keep the interrupt latency brief, only absolutely indispensable actions should be effected in the interrupt service routine, which ensures reception of the interrupt and trigger its (post) processing within a task. Other processing levels are: Operating system level and task level. |
| Network | A network defines the assignment (1:N) between a logical communication grouping and a physical layer on which different modules are connected to (either CAN or LIN). One network consists of one channel, Y networks consists of 1..Z channel(s). We say network if we talk about more than one bus. |
| Physical layer | An electrical circuit that connects an ECU to a communication media. |
| Platform | The sum of micro controller derivative, communication controller implementation and compiler is called platform. |
| Post-build | This type of configuration is possible after building the software module or the ECU software. The software may either receive parameters of its configuration during the download of the complete ECU software resulting from the linkage of the code, or it may receive its configuration file that can be downloaded to the ECU separately, avoiding a re-compilation and re-build of the ECU software modules. In order to make the post-build time re-configuration possible, the re-configurable parameters shall be stored at a known memory location of ECU storage area. |
| Segmented data transfer | See segmented communication |
| Software architecture | A software architecture is the structure or structures of a system, which comprises software components, the external visible properties of these components and the relationships among them. |
| Transport Protocol | Some information that must be transferred over the CAN/LIN bus does |

| | |
|---|---|
| | not fit into individual message frames because the data length exceeds the maximum of 8 bytes. In this case, the sender must divide up the data into a number of messages. Additional information is necessary for the receiver to put the data together again. |

Table 8-1 Glossary

| | |
|---|---|
| API | Application Programming Interface |
| AUTOSAR | Automotive Open System Architecture |
| BS | Block Size |
| BSW | Basis Software |
| BSWMD | Basic Software Module Description |
| CAN | Controller Area Network protocol originally defined for use as a communication network for control applications in vehicles. |
| CANTP | CAN Transport Layer |
| CF | Consecutive Frame |
| COM | Communication |
| CTS | Clear To Send |
| DCM | Diagnostic Communication Manager |
| DEM | Diagnostic Event Manager |
| DET | Development Error Tracer |
| DL | Data Length |
| DLC | Data Length Code, Number of data bytes of a CAN message |
| ECU | Electronic Control Unit |
| EXTENDED | Addressing type |
| FC | Flow Control |
| FF | First Frame |
| FFFF | undefined |
| FS | Flow Status Control |
| HIS | Hersteller Initiative Software |
| ID | Identifier |
| ISO | International Standardization Organization |
| LIN | Local Interconnect Network |
| MICROSAR | Microcontroller Open System Architecture (the Vector AUTOSAR solution) |
| OBD | undefined |

| | |
|---|---|
| OEM | Original Equipment Manufacturer |
| OS | Operating System |
| OVFLW | Overflow |
| PDU | Protocol data unit |
| PDUR | PDU Router |
| ROM | Read-Only Memory |
| RTE | Runtime Environment |
| SA | Source Address |
| SDU | undefined |
| SF | Single Frame |
| SN | Sequence Number |
| SRS | Software Requirement Specification |
| ST | Separation Time |
| STANDARD | Addressing type |
| SWC | Software Component |
| SWS | Software Specification |
| TA | Target Address |
| TP | Transport Protocol |
| TPCI | Transport Protocol Control Information |
| USDT | Unacknowledged Segmented Data Transfer |
| UUDT | Unacknowledged Unsegmented Data Transfer |
| WT | Wait |

Table 8-2 Abbreviations

Visit our website for more information on

> News
> Products
> Demo software
> Support
> Training data
> Addresses