VECTOR >

# MICROSAR CAN Transport Layer
Technical Reference

Version 6.00.00

| Authors | visdth, visath, visore, viswnk |
|---------|-------------------------------|
| Status  | Released                      |

 

| visath | 2019-03-21 | 3.03.00 | > Mixed29 and normal fixed addressing modes<br>> Generic connections using metadata |
|---|---|---|---|
| visath | 2020-08-21 | 4.00.00 | > Minor fixes |
| viswnk, visath | 2021-06-07 | 5.00.00 | > Custom Addressing<br>> Simultaneous generic connections over shared N-PDUs. |
| visore | 2022-09-06 | 5.01.00 | > Non-Strict Length Checks |
| visore | 2022-12-06 | 5.02.00 | > Failure Indication Callouts |
| viswnk | 2022-10-19 | 6.00.00 | > Multi Partition Support |

## Reference Documents

| No. | Source | Title | Version |
|---|---|---|---|
| [1] | AUTOSAR | AUTOSAR_SWS_CANTransportLayer.pdf | 4.0.0 |
| [2] | AUTOSAR | AUTOSAR_SWS_CANTransportLayer.pdf | 4.2.2 |
| [3] | AUTOSAR | AUTOSAR_SWS_CANInterface.pdf | 5.0.0 |
| [4] | AUTOSAR | AUTOSAR_SWS_PDURouter.pdf | 3.2.0 |
| [5] | ISO | ISO 15765-2; Road vehicles — Diagnostics over CAN — Part 2: Network layer services | 2016 |
| [6] | AUTOSAR | AUTOSAR_SWS_DevelopmentErrorTracer.pdf | 3.2.0 |
| [7] | Vector | TechnicalReference_Asr_Dbg.pdf | 1.0.0 |
| [8] | Vector | TechnicalReference_PostBuildLoadable.pdf | 1.2.0 |
| [9] | Vector | TechnicalReference_IdentityManager.pdf | 1.0.0 |
| [10] | Vector | TechnicalReference_Det.pdf | 2.0.4 |

**Caution**
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.

# Contents

# 1   Introduction

This document describes the functionality, API and configuration of the AUTOSAR BSW module CanTp as specified in [1] and [2].

| Supported AUTOSAR Release: | 4 | |
|---|---|---|
| Supported Configuration Variants: | pre-compile, post-build-loadable, post-build-selectable | |
| Vendor ID: | CANTP_VENDOR_ID | 30 decimal (= Vector-Informatik, according to HIS) |
| Module ID: | CANTP_MODULE_ID | 35 decimal (according to ref. [1]) |

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 [5]) 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.

## 1.1 Architecture Overview

The next figure shows the interfaces to adjacent modules of the CanTp. These interfaces are described in chapter 4.
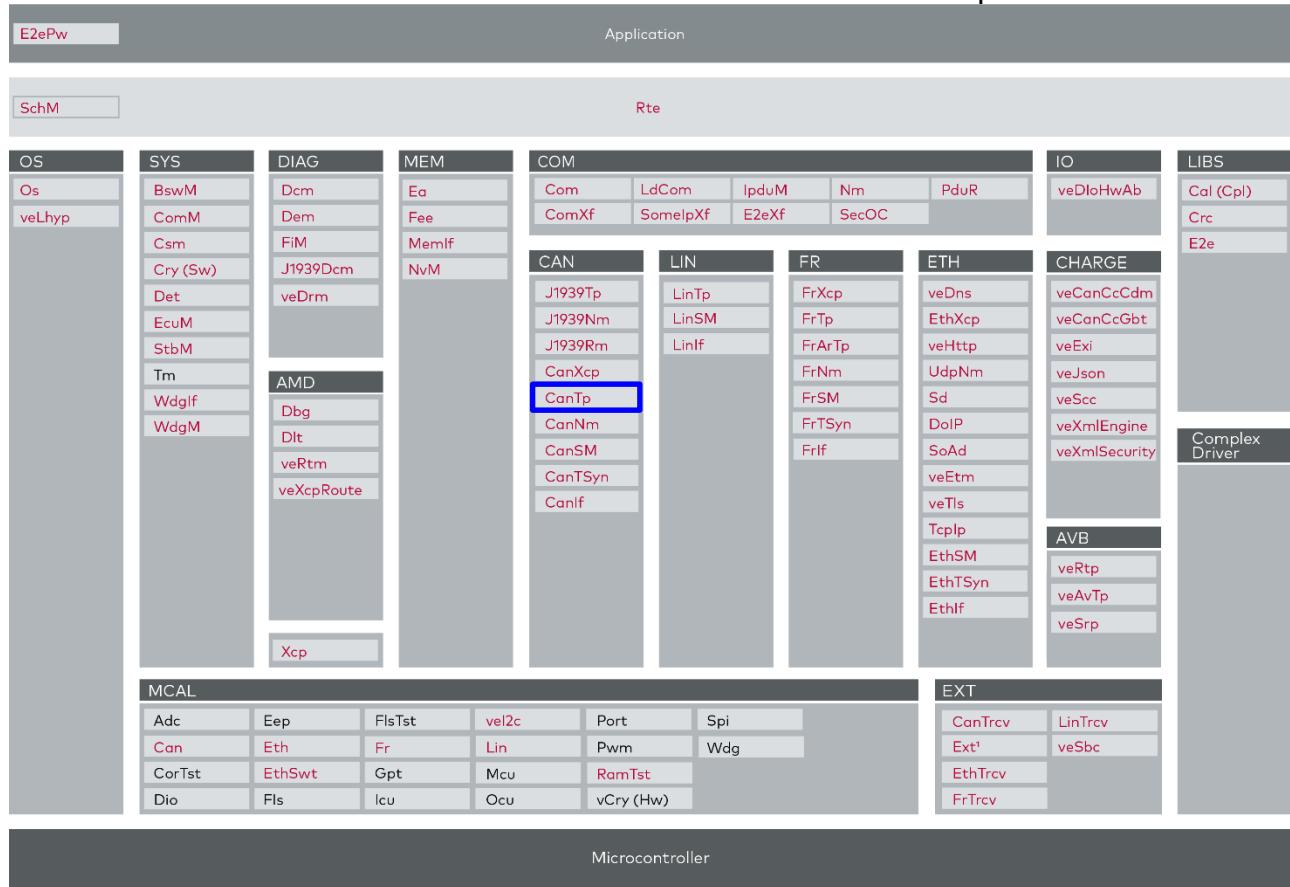


Figure 1-1    AUTOSAR 4.2 Architecture Overview
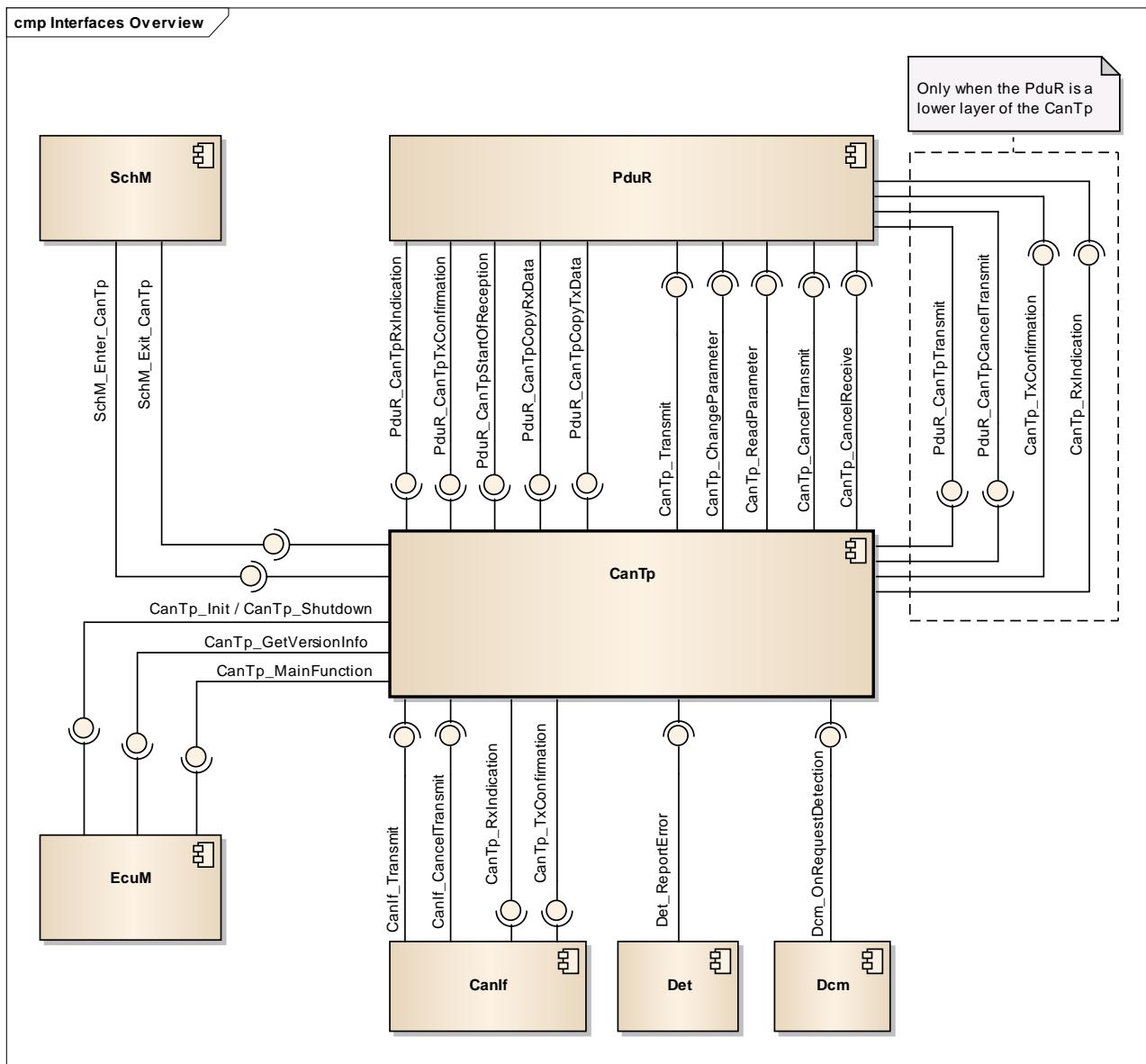
Figure 1-2    Interfaces to adjacent modules of the CanTp

# 2 Functional Description

## 2.1 Features

The features listed in the following tables cover the complete functionality specified for the CanTp.

The AUTOSAR standard functionality is specified in [1], the corresponding features are listed in the tables

> Table 2-1   Supported AUTOSAR standard conform features

> Table 2-2   Not supported AUTOSAR standard conform features

Vector Informatik provides further CanTp functionality beyond the AUTOSAR standard. The corresponding features are listed in the table

> Table 2-3   Features provided beyond the AUTOSAR standard

The following features specified in [1] and [2] are supported:

| Supported AUTOSAR Standard Conform Features |
| --- |
| Segmented and unsegmented data transmission |
| Segmented and unsegmented data reception |
| Control of data flow |
| Supervision of timeouts |
| Detection of errors during segemented communication |
| Transmission cancellation |
| Reception cancellation |
| Post-Build Loadable |
| MICROSAR Identity Manager using Post-Build Selectable |
| Generic connections using metadata (AUTOSAR 4.2.2) |

Table 2-1      Supported AUTOSAR standard conform features

## 2.1.1 Deviations

The following features specified in [1] and [2] are not or only partly supported:

| Description | ASR Version |
| --- | --- |
| CanTp_Transmit: For robustness reasons (e.g. in case of delayed buffer provision), the CanTp internally stores the frames to be transmitted. | 4.0.3 |
| CanTp_ GetVersionInfo: The function is always implemented as a regular function. | 4.0.3 |

| Description | ASR Version |
|---|---|
| Partially-generic connections are not supported. For example, a generic connection with normal fixed addressing and a static target address but dynamic source address is not supported. | 4.2.2 |
| Mixed29 and normal fixed addressing: The source address (N_SA) and target address (N_TA) are set by the CanTp only for generic connections. For non-generic connections, the CAN IDs configured for the PDUs in the CanIf are used (as with standard addressing). | 4.2.2 |

Table 2-2    Not supported AUTOSAR standard conform features

## 2.1.2 Additions/ Extensions

The following features are provided beyond the AUTOSAR standard:

| Features Provided Beyond The AUTOSAR Standard |
|---|
| Split MainFunction |
| Notification of Failed Buffer Request |
| Handling of FC Frames with a Reserved STmin |
| Dynamic Channel Assignment |
| Single Buffer Optimization |
| Transmit Queue |
| Synchronous Transmission |
| CAN-FD Support |
| Separation Time by Application |
| Mixed11 address extension forwarding |
| Support of the PduR as lower layer |
| Custom addressing |
| Non-strict length checks |
| Multi Partition Support |

Table 2-3    Features provided beyond the AUTOSAR standard

### 2.1.2.1 Split CanTp_MainFunction

In extension to the CanTp SWS [1] the Vector CanTp supports two additional API functions (`CanTp_MainFunctionRx` and `CanTp_MainFunctionTx`) in case a split main function is configured. 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.

Configuration attribute: CanTpEnableSplitMainFunction

### 2.1.2.2 Notification of Failed Buffer Request

If the call to `CanTpStartOfReception` returns BUFREQ_E_NOT_OK or BUFREQ_E_OVFL, according to AUTOSAR the connection shall be terminated. Additionally each terminated reception shall be notified to the upper layer. However, in the mentioned

case the upper layer already rejected the reception by returning an invalid buffer status. Therefore no additional notification by the CanTp may be required.

In the Vector CanTp, this behavior is configurable. If the related switch is activated, the notification is only performed if at least one valid buffer (BUFREQ_OK or BUFREQ_E_BUSY) have been provided.

Configuration attribute: CanTpOnlyNotifyInformedAppl

### 2.1.2.3 Handling of FC Frames with a Reserved STmin

When using the standard implementation according to ISO 157675-2, the CanTp must accept reserved STmin values (0x80 ... 0xF0, 0xFA ... 0xFF); the connection is then processed with the slowest value for STmin (127msec).

In the CanTp, it can be configured to cancel the transmission instead.

Configuration attribute:  CanTpRejectFcWithReservedStMin

### 2.1.2.4 Dynamic and Static BlockSize and STmin

In AUTOSAR 4 and in older ISO specifications, the block size and the STmin values of the first Flow Control shall be used throughout the whole connection (CANTP067).

ISO 15765-2:2016 ([5]) supports both dynamic and static flow control parameters. In the CanTp, the desired behavior can be configured as follows:

> For Rx connections, the block size can be configured to be constant until the end of the reception. It is then calculated once after receiving the FF, based on the configured maximum block size and the available buffer (default). When using the alternative behavior, the block size is re-calculated whenever a Flow Control have to be transmitted. For STmin, always the configured value is used.
Configuration attribute: CanTpEnableConstantBS

> For Tx connections it can be configured if the CanTp shall either evaluate only the first or all Flow Control frames.
Configuration attribute: CanTpUseOnlyFirstFc

### 2.1.2.5 Dynamic Channel Assignment

According to AUTOSAR, each N-SDU shall be assigned statically to one connection channel. Since not always all N-SDUs need to be processed simultaneously, the CanTp provides the possibility to limit the number of channels that can be used in parallel. In this case, N-SDUs are dynamically assigned to a channel during runtime. If no channel is available, the reception / transmission is rejected.

This reduces resource consumption, as the memory needed to handle reception or transmission is shared between multiple N-SDUs. However, it adds a little runtime overhead for channel management.

Configuration attribute: CanTpDynamicChannelAssignment

### 2.1.2.6 Single Buffer Optimization

According to AUTOSAR, the CanTp must include a handling for segmented receive buffers, i.e. it is allowed that CanTpStartOfReception returns a smaller buffer than overall data

length specified in the first frame. The CanTp will then request an additional buffer during reception.

For diagnostic applications, often only one buffer is provided at the beginning where its size is sufficient to store the complete message. Therefore no additional buffer must be requested by the CanTp.

If it can be ensured that always only one single buffer is used, the CanTp provides a configuration switch to remove the code to handle segmented buffers. This will reduce size and runtime.

Configuration attributes:  CanTpOptimizeSingleRxBuffer

### 2.1.2.7    Transmit Queue

If an N-PDU is used by different connections and more than one connection requests the CanIf to transmit this N-PDU at the same time, the subsequent TxConfirmation cannot be uniquely assigned to the correct connection.

There may be two reasons why an N-PDU is used by different connections:

> A channel is configured as full duplex: the conflict occurs here if the Rx connection tries to transmit an FC while the Tx connection is transmitting SF, FF or CFs.

> Extended, mixed11 or mixed29 addressing is used: then one N-PDU can be used by multiple channels, as the addressing information is part of the protocol data. However, this protocol data is not available when the TxConfirmation is processed.

Currently AUTOSAR does not describe how to handle these cases. By default the CanTp tracks for each N-PDU if a transmission is in progress or not. If a channel tries to transmit an N-PDU that is in use by another N-SDU, the request is rejected and the respective channel will retry transmission on task level.

However, the more often such conflicts occur (e.g. if many N-SDUs with extended addressing use the same N-PDU), the higher is the risk of sporadically unexpected behavior like connection timeouts due to the delay caused by the retry on task level. To eliminate this risk and to optimize the throughput performance, the CanTp can be configured to queue the transmit requests to the CanIf. Entries in the queue are transmitted as soon as the transmission of the previous N-PDU is completed.

The main drawback of this feature is increased memory consumption.

Configuration attributes: CanTpEnableTransmitQueue

**Note**
The default transmit queue size is 4. To set a different queue size, the following line must be added to a user config file:

```
#define CANTP_TX_QUEUE_SIZE <new size>
```

Please note that due to implementation reasons, only queue sizes with a power of two are allowed (2, 4, 8, 16…).

## 2.1.2.8 Asynchronous and Synchronous behavior of CanTp_Transmit

By default, the API CanTp_Transmit is asynchronous. This means it only prepares the connection, while the request for the payload data to the upper layer and the transmission of the first CAN frame will be done during the next task cycle.
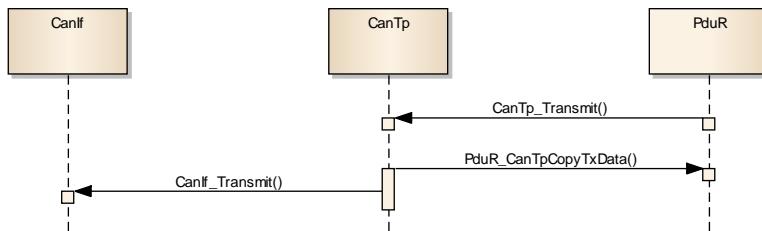


Figure 2-1  Sequence Diagram: Asynchronous Transmission

The CanTp can be configured to make CanTp_Transmit synchronous. Then the payload request to the upper layer and the transmit request to the CanIf are done in the context of CanTp_Transmit. This will slightly improve transmission speed, but requires also that the upper layer is able to handle calls to the CopyTxData function before CanTp_Transmit returns.
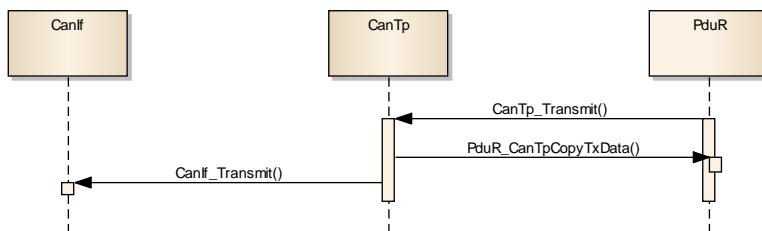


Figure 2-2  Sequence Diagram: Synchronous Transmission

Configuration attributes: CanTpEnableSynchronousTransmit

**Caution**
Synchronous Transmission might not be supported by non-MICROSAR upper layers.

## 2.1.2.9 Support of PduR Interface according to AUTOSAR 4.1.2

Although the CanTp is not yet fully compliant with AUTOSAR 4.1.2, it already supports the PduR API signatures according to the updated revision.

The following changes have been made by AUTOSAR in the interface between CanTp and PduR:

> `PduR_CanTpStartOfReception`: pointer to PduInfoType as additional parameter added for meta data support (currently not supported by CanTp; will be set to NULL)

> `PduR_CanTpRxIndication`, `PduR_CanTpTxConfirmation`: type of the Result parameter changed from NotifResultType to Std_ReturnType

| PduR API Changes |
| --- |
| AUTOSAR 4.0.3 |
| `BufReq_ReturnType PduR_CanTpStartOfReception( PduIdType id,`<br>`                                               PduLengthType TpSduLength,`<br>`                                               PduLengthType* bufferSizePtr );` |
| `void PduR_CanTpRxIndication( PduIdType CanTpRxPduId,`<br>`                             NotifResultType Result );` |
| `void PduR_CanTpTxConfirmation( PduIdType CanTpTxPduId,`<br>`                               NotifResultType Result );` |
| AUTOSAR 4.1.2 |
| `BufReq_ReturnType PduR_CanTpStartOfReception( PduIdType id,`<br>`                                               PduInfoType * PduInfoPtr,`<br>`                                               PduLengthType TpSduLength,`<br>`                                               PduLengthType* bufferSizePtr );` |
| `void PduR_CanTpRxIndication( PduIdType CanTpRxPduId,`<br>`                             Std_ReturnType Result );` |
| `void PduR_CanTpTxConfirmation(PduIdType CanTpTxPduId,`<br>`                               Std_ReturnType Result );` |

Table 2-4     PduR API changes between AR4.0.3 and 4.1.2

With a Vector PduR, the DaVinci Configurator Pro 5 will automatically detect the PduR version and activate the appropriate API signature.

With a Non-Vector PduR in the stack, the AUTOSAR version must be set by providing one of the following definitions (e.g. via compiler config):

AUTOSAR 4.0.3: `#define MSR_PDUR_API_AR_VERSION   0x403`

AUTOSAR 4.1.2: `#define MSR_PDUR_API_AR_VERSION   0x412`

### 2.1.2.10   CAN-FD Support

The CanTp supports the ISO 15765-2:2016, which not only introduces CAN-FD frames with more than 8 byte payload, but also supports segmented transfers with more than 4095 byte.

#### 2.1.2.10.1  CAN Messages with more than 8 Byte

CAN-FD frames support a higher bit rate in the data field of a CAN frame, as well as a maximum length of up to 64 byte per frame. For the CanTp behavior, only the data length is relevant, CAN-FD frames with only 8 byte are treated the same way as classic CAN frames.

The maximum possible CAN DLC is configured by the PduLength parameter of a global PDU in the ECUC:

   /EcuC/EcucPduCollection/Pdu/PduLength

To use CAN-FD for an N-SDU, all of the following configuration settings are required:

> CanTpRxNPduRef / CanTpTxNPduRef must reference a global PDU with a PduLength > 8

> > CanTpRxTaType / CanTpTxTaType must be set to `CANTP_CANFD_FUNCTIONAL` or `CANTP_CANFD_PHYSICAL`

> > The global switch CanTpFlexibleDataRateSupport must be enabled

The flow control N-PDU references are not taken into account, as these frames always need less than 8 byte. It does not affect the CanTp behavior whether if they are configured in the CanIf as CAN-FD or as classic CAN.

> **Reference**
> For restrictions of the N-PDU usage on a channel, please also refer to chapter 2.1.3.2.

### 2.1.2.10.2 CAN-FD Frame Padding

With CAN-FD, not all DLCs between 8 and 64 are valid. A CanTp frame must always be padded to the next valid DLC. The byte value which is used to fill up the frame is either random or user defined, depending on the following configuration parameters:

CanTp/CanTpGeneral/CanTpHavePaddingByte

CanTp/CanTpGeneral/CanTpPaddingByte

The PaddingActivation setting, which can be configured globally and N-SDU specific, only applies for frames which require a DLC less than 8 byte.

> **Example**
> > If a CanTp CAN-FD frame (PCI + payload) needs 6 byte, it is padded to 8 byte if padding is active, but left at 6 byte if padding is disabled.
> > If a CanTp CAN-FD frame (PCI + payload) needs 21 byte, it is always extended to 24 byte, which is the next valid CAN-FD length

### 2.1.2.10.3 Segmented Messages with more than 4095 Byte

ISO15765-2:2016 also introduced an extended first frame definition (in the following referred to as "long first frame"; LFF), which uses a data length of 32 bit. For backward compatibility, LFFs are only used if the overall message length is above 4095 byte. Otherwise, the standard first frame format with 12 bit data length is used.

As the LFF does not depend on the length of the used N-PDUs, it can also be used with classic CAN frames. However, for ISO compatibility it is recommended to enable this functionality when using CAN-FD.

Configuration attribute: CanTpSupportLongFirstFrames

> **Caution**
>
> In the ECUC module, the type of the global PduLength can be configured:
>
> /EcuC/EcucPduCollection/PduLengthTypeEnum
>
> To use the full 32 bit data length of the LFF, it must be set to UINT32.
>
> If set to UINT16, the maximum data length will be limited to 65535 byte and the CanTp will reject received LFFs with a longer data length.
>
> For transmission requests, an overrun can't be detected by the CanTp as the compiler will already truncate the data length passed to CanTp_Transmit.

### 2.1.2.11 Separation Time by Application

The accuracy of the STmin calculated by the CanTp depends on its task cycle. If STmin values are required which are in the range or below the CanTp task cycle time, this may not be acceptable.

One solution may be to reduce the task cycle time. However, this is usually not satisfying since it produces too high CPU load. An external timer (like in the OS or in hardware) can be an alternative.

For this, the CanTp provides an optional call-out which notifies the application whenever STmin need to be started. By the return value of the notification function, the application can indicate whether to do STmin handling by itself, or leave it to the CanTp.

If the application accepts to handle the separation time, it has to set up a timer and call `CanTp_StopSeparationTime()` when the timer expired. This will trigger the transmission of the next CF.

It is allowed to call `CanTp_StopSeparationTime()` anytime between the call to `Appl_StartSeparationTime()` and before the end of the configured N_Cs time. Only if N_Cs expires and the call-back has not been called yet, the CanTp will send the next CF by itself to fulfill the ISO15765-2 performance requirement (see 2.8.2.2). Calling `CanTp_StopSeparationTime()` afterwards has no effect.
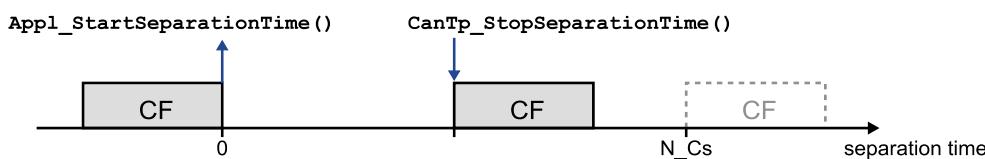


Figure 2-3    Separation Time by Application

To activate the feature, the call-out function name must be specified in the config tool:

CanTp/CanTpGeneral/CanTpApplSTminStartFunction

### 2.1.2.12 Mixed11 address extension forwarding

If this feature is enabled, the address extension (N_AE) of each first frame and single frame received by the CanTp is forwarded to the Dcm. The forwarding is done using the call-out function

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

where `CanTpRxPduId` is the Id of the received Rx N-PDU and `N_AE` is the address extension. The prototype of this function is expected to be provided in the Dcm_Cbk.h header file.

Configuration attribute: CanTpEnableDcmRequestDetect

> **Note**
> The forwarding is done for all the first frames and single frames configured with mixed11 addressing. Even the N_AE of frames discarded by the CanTp (because of an unrecognized N_AE) will be forwarded.

### 2.1.2.13 Support of the PduR as lower layer

The CanTp transmits and receives N-PDUs using the functions offered by the CanIf (as defined by AUTOSAR), but it can also be configured to use the PduR's communication interface instead. Since the lower layer is assigned per channel, it's possible to configure the CanTp to communicate over both the CanIf and the PduR.

Configuration attributes: CanTpSupportedLowerLayers, CanTpChannelLowerLayer

> **Note**
> In most parts of this document the CanIf is assumed to be the only lower layer (as defined by AUTOSAR), but all references to the CanIf and its functions can be replaced by the PduR and its communication interface when this feature is active.

### 2.1.2.14 Generic connections with metadata

According to AUTOSAR, connections whose addressing information (N_AI) is determined dynamically at runtime are called generic connections. The dynamic addressing information is shared between the modules using metadata.

Depending on the addressing mode, the CanTp and its lower/upper layers will use metadata to share the Source Address (N_SA), the Target Address (N_TA) and the Address Extension (N_AE) of an ongoing connection.

Configuration attributes: CanTpGenericConnectionSupport, CanTpDynIdSupport.

### 2.1.2.15 Custom addressing

With the help of metadata and callout functions, an addressing format different from the ones considered by AUTOSAR can be used for CanTp communication.

The provided callout functions are used by the CanTp to interpret the metadata of received N-PDUs and transmitted N-SDUs. The information returned by the callout functions is used by the CanTp to identify the CAN frames associated to a connection, and to send data and flow control frames with the correct CAN ID.

Configuration attributes: See section 5.3.5

### 2.1.2.16 Non-strict length checks

By default, the CanTp ignores the reception of SFs if the value of SF_DL does not fall into the valid range defined in the ISO 15765-2:2016. According to the ISO the CanTp also ignores the last CF, if the received CAN_DL does not match the expected RX_DL when verifying the received CAN frames.

The CanTp can be configured to accept SFs and last CF frames where CAN_DL has at least the minimum value, i.e. CAN_DL values larger than expected are also considered to be valid.

Configuration attribute: CanTpEnableNonStrictLengthCheck

### 2.1.3 Limitations

### 2.1.3.1 Memory Optimization

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.

### 2.1.3.2 Channel Assignment

In the Vector CanTp, a channel always consists of at most one Rx N-SDU and one Tx N-SDU. Furthermore, to each channel at most one Rx N-PDU and one Tx N-PDU is assigned. These N-PDUs are shared between the two N-SDUs as shown in Figure 2-4.

As with CAN-FD an N-PDU can either be CAN2.0 or CAN-FD but not both, this implies that you have to use a CAN-FD Flow Control if the SFs / FFs / CFs of the opposite direction use CAN-FD. Otherwise you will need separate channels for the Rx and Tx N-SDUs.
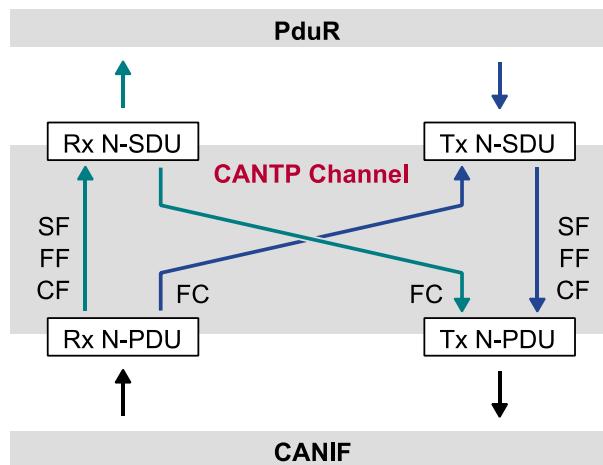


Figure 2-4    Channel Assignment for N-SDUs and N-PDUs

While each N-SDU represents a CanTp connection, the N-PDUs identify the CAN frames that are transmitted and received via the CanIf.

With standard and normal fixed addressing, each N-PDU can only be assigned to one channel. With extended, mixed11 and mixed29 addressing, an N-PDU may be used on multiple channels, but only with different address information (N_TA / N_AE). An N-PDU may also be used on multiple channels containing generic connections.

### 2.1.3.3 Channel Addressing

Usage of different addressing within the same channel is not allowed.

The addressing format – CANTP_STANDARD, CANTP_EXTENDED, CANTP_MIXED11, CANTP_MIXED29 or CANTP_NORMALFIXED – must be identical for each pair of Rx/Tx N-SDUs assigned to the same channel.

### 2.1.3.4 Data Length Parameter

According to the AUTOSAR BSWMD specification, each N-SDU has a Data Length parameter (CanTpRxDl, CanTpTxDl). However, it is not clearly specified if this data length applies to the N-SDU (complete message length) or to the N-PDU (CAN message data length). As furthermore the parameters are anyway deprecated in AR 4.1.1 (see also AUTOSAR RFC 53101), they are no longer evaluated by the CanTp.

For compatibility reasons, the parameters are part of the ECUC file, but their values have no effect.

## 2.2 Initialization

Independent of the "standard" or "Multi Partition" use case, the initialization sequence is uniform and similar to most of the components of the MICROSAR communication stack. In order to be independent of the initialization sequence of particular OS Applications:

> the "one and only" Configuration Set pointed to by the given ConfigPtr is passed to the CanTp by the ECUs startup code using the API `CanTp_PreInit()`.

> the prototype of the API function `CanTp_Init()` remains unchanged but the ConfigPtr parameter, passed to that API, is ignored during runtime.

Depending on the use case, `CanTp_Init()` is called once for the ECU's only application, or multiply, for all OS Applications of the ECU. `CanTp_Init()` asserts that `CanTp_PreInit()` has been called, before, and uses the given Configuration Set to initialize all global variables of the CAN Transport Layer and brings it to the state "idle", in which reception and transmission tasks can be started.

## 2.3 States

The CanTp module has two internal states, `CANTP_OFF` and `CANTP_ON`. After power up, the CanTp is in the `CANTP_OFF` state. In this state no communication tasks can be performed.

The CanTp changes to the internal state `CANTP_ON` when it has been successfully initialized. Segmentation and reassembly tasks are only performed when the module is in this state. After initialization, all transport protocol connections are in a sub-state of `CANTP_ON`, in which neither transmission nor reception is in progress (`CANTP_RX_IDLE` and `CANTP_TX_IDLE`).

If called when the CanTp module is in the global state `CANTP_ON,` the function `CanTp_Init` returns the module to state Idle and all current connections are terminated.

The function `CanTp_Shutdown` stops the CanTp module properly and sets the global state to CANTP_OFF.
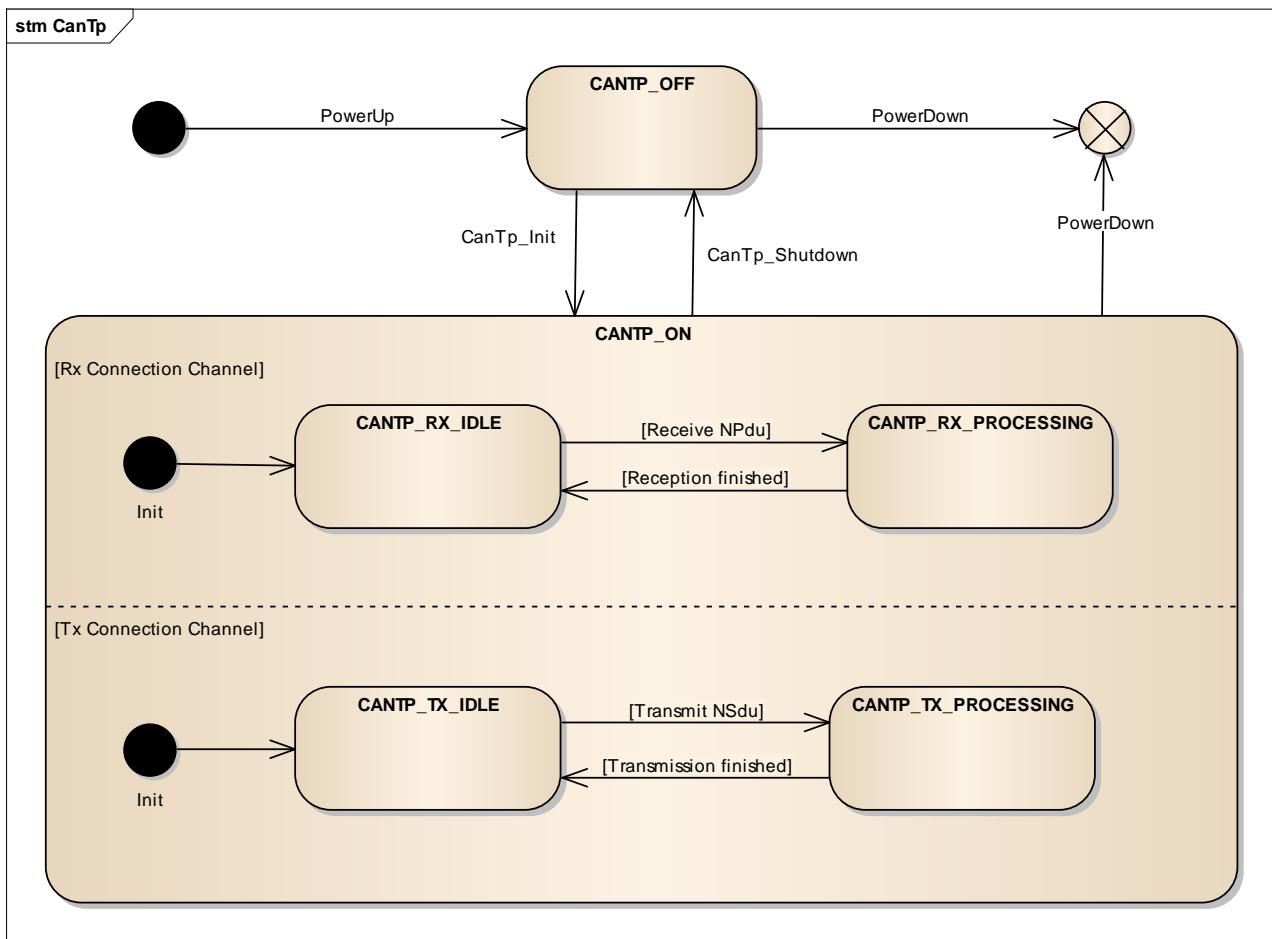
Figure 2-5    CanTp internal states

## 2.4    Main Functions

The CanTp_MainFunction controls the timing behavior of the CanTp and performs all tasks that have to be done cyclically.

Especially for the calculation of the different timings (delays, timeouts) it is important to call the main function periodically with the time interval that have been configured with the 'CanTpMainFunctionPeriod' parameter in the generation tool.

It is also possible to separate the main function into Rx and Tx, e.g. to optimize the throughput (see chapter 2.1.2.1 for details).

## 2.5    Error Handling

### 2.5.1    Development Error Reporting

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 identifies the services which are described in 4.1. The following table presents the service IDs and the related services. Services marked with an asterisk (*) are internal service IDs:

| Service ID | Service |
|---|---|
| 0x01 | CanTp_Init |
| 0x02 | CanTp_Shutdown |
| 0x03 | CanTp_Transmit |
| 0x04 | CanTp_RxIndication |
| 0x05 | CanTp_TxConfirmation |
| 0x06 | CanTp_MainFunction |
| 0x07 | CanTp_GetVersionInfo |
| 0x08 | CanTp_CancelTransmit |
| 0x09 | CanTp_CancelReceive |
| 0x0A | CanTp_ChangeParameter |
| 0x0B | CanTp_ReadParameter |
| 0x20* | CanTp_MainFunctionRx |
| 0x21* | CanTp_MainFunctionTx |
| 0x30* | CanTp_RxGetBuffer |
| 0x31* | CanTp_TxGetBuffer |
| 0x32* | CanTp_RxTransmitFrame |
| 0x33* | CanTp_TxTransmitFrame |
| 0x34* | CanTp_RxInit |
| 0x35* | CanTp_TxInit |
| 0x36* | CanTp_StopSeparationTimer |

Table 2-5      Service IDs

The errors reported to DET are described in the following table. Errors marked with an asterisk (*) are Vector specific:

| Error Code | | Description |
|---|---|---|
| 0x01 | CANTP_E_PARAM_CONFIG | API called with wrong parameters |
| 0x02 | CANTP_E_PARAM_ID | API called with wrong parameters |
| 0x03 | CANTP_E_PARAM_POINTER | API called with wrong parameters |
| 0x20 | CANTP_E_UNINIT | API service used without module initialization |
| 0x30 | CANTP_E_INVALID_TX_ID | Invalid Transmit N-PDU identifier |
| 0x40 | CANTP_E_INVALID_RX_ID | Invalid Receive N-PDU identifier |
| 0x50 | CANTP_E_INVALID_TX_BUFFER | Invalid Transmit buffer provided |
| 0x60 | CANTP_E_INVALID_RX_BUFFER | Invalid Receive buffer provided |
| 0x70 | CANTP_E_INVALID_TX_LENGTH | Invalid data length of the transmit N-PDU |

| Error Code | | Description |
|---|---|---|
| 0x80 | CANTP_E_INVALID_RX_LENGTH | Invalid data length of the receive N-PDU |
| 0x90 | CANTP_E_INVALID_TA_TYPE | Functional FF received, or transmission request for a functional N-SDU with SF data length |
| 0xA0 | CANTP_E_OPER_NOT_SUPPORTED | Requested operation is currently not available (e.g. cancel a transmission that is not in progress) |
| 0xB0 | CANTP_E_COM | Implementation specific error |
| 0xB1 | CANTP_E_INVALID_RX_STATE* | Rx state machine is in an invalid state |
| 0xB2 | CANTP_E_INVALID_TX_STATE* | Tx state machine is in an invalid state |
| 0xB3 | CANTP_E_INVALID_FRAME_TYPE* | An invalid frame type occurred |
| 0xC0 | CANTP_E_RX_COM | General reception error |
| 0xC1 | CANTP_E_RX_TIMEOUT_AR* | N_Ar timeout occurred |
| 0xC2 | CANTP_E_RX_TIMEOUT_BR* | N_Br timeout occurred |
| 0xC3 | CANTP_E_RX_TIMEOUT_CR* | N_Cr timeout occurred |
| 0xC4 | CANTP_E_RX_INVALID_SN* | CF with invalid sequence number received |
| 0xC5 | CANTP_E_RX_WFTMAX* | Max. number of wait frames transmitted |
| 0xC6 | CANTP_E_RX_RESTART* | Connection terminate due to new SF/FF reception |
| 0xC7 | CANTP_E_RX_TRANSMIT_ERROR* | Transmission of a flow control frame failed |
| 0xD0 | CANTP_E_TX_COM | General transmission error |
| 0xD1 | CANTP_E_TX_TIMEOUT_AS* | N_As timeout occurred |
| 0xD2 | CANTP_E_TX_TIMEOUT_BS* | N_Bs timeout occurred |
| 0xD3 | CANTP_E_TX_TIMEOUT_CS* | N_Cs timeout occurred |
| 0xD4 | CANTP_E_TX_FC_OVFL* | FC.OVFL received |
| 0xD5 | CANTP_E_TX_INVALID_FS* | FC with invalid flow status received |
| 0xD6 | CANTP_E_TX_RES_STMIN* | FC with reserved STmin received, but not allowed |
| 0xD7 | CANTP_E_TX_TRANSMIT_ERROR* | Transmission of a frame failed |

Table 2-6      Errors reported to DET

**FAQ: How to avoid DET errors for failed CanTp communication?**

AUTOSAR specifies that the CanTp shall not only report a DET for typical integration errors, but also for communication errors like timeouts. Although sometimes helpful, in most cases this is not desired.

To suppress reporting a DET error for all erroneously terminated connections, filters with the following parameters have to be configured in the DET module:

> moduleId   = 0x23
> instanceId = 0x00
> apiId       = 0x34 and 0x35
> errorId     = 0xFF

Please note that the filtering mechanism is a Vector specific feature, which is described in more detail in the Vector DET Technical Reference.

When using other DET implementations, these errors can be filtered in a DET callout which abandons further DET processing and continues the CanTp code.

### 2.5.1.1 Parameter Checking

AUTOSAR requires that API functions check the validity of their parameters. The checks in Table 2-7 are internal parameter checks of the API functions. These checks are for development error reporting and can be en-/disabled via the configuration parameter `CanTp_DEV_ERROR_DETECT`.

The following table shows the parameter checks performed by the CanTp services. For parameter checks marked with an asterisk (*), only the error reporting can be deactivated. The check will also be performed if development error reporting is disabled.

| Service \ Check | CanTpRxSduId | CanTpTxSduId | CanTpRxPduId | CanIfTxPduId | pCanTpRxPduPtr | pData | Pid | Pval | pVersionInfo |
|---|---|---|---|---|---|---|---|---|---|
| CanTp_RxIndication | | | ■ | | ■ | | | | |
| CanTp_CancelReceive | ■* | | | | | | | | |
| CanTp_ChangeParameter | ■* | | | | | | ■* | ■* | |
| CanTp_ReadParameter | ■* | | | | | | ■* | ■* | |
| CanTp_Transmit | | ■* | | | | ■ | | | |
| CanTp_CancelTransmit | | ■* | | | | | | | |
| CanTp_TxConfirmation | | | | ■ | | | | | |
| CanTp_GetVersionInfo | | | | | | | | | ■ |
| CanTp_StopSeparationTime | | ■* | | | | | | | |

Table 2-7    Development Error Reporting: Assignment of checks to services

### 2.5.2 Production Code Error Reporting

In AR4, the CanTp reports no production errors to the Diagnostic Event Manager.

## 2.6 Channel Mode

In the configuration tool, two connections can be grouped to one logical channel (see also 2.1.3.2).

For each of these logical channels it can be configured if the channel is half or full duplex.

> Half Duplex: only the Rx or the Tx N-SDU of a channel can be active at a time. A connection which is started while the opposite direction is being processed will be rejected.

> Full Duplex: bidirectional communication is possible at any time.

When only one N-SDU is configured for a channel, the channel mode parameter has no effect.

## 2.7 Connection Channels

A connection channel represents an internal path for transmission or reception of an N-SDU during runtime. It uses its own resources such as internal buffer, timer or state machine. Therefore, each connection channel is independent from the others. The connection channels are only for CanTp internal use and not accessible externally.

By default, each N-SDU is statically linked to one connection channel (exception: dynamic channel assignment is used; see 2.1.2.5).

The CanTp is able to manage several connections for different N-SDUs simultaneously. However it is not possible to handle two receptions or two transmissions with the same N-SDU identifier in parallel.

If a new FF / SF is received for an already active N-SDU, the connection is terminated and restarted.

If a new transmission request is started for an already active N-SDU, the request is rejected. If it is required to start a new transmission before the previous one is finished, the active connection must first be terminated by using the transmit cancellation function (see 4.1.11 CanTp_CancelTransmit()).

## 2.8 Connection Timings

### 2.8.1 Timing Parameters

The ISO specifies several protocol timing parameters for receiver and transmitter side. The following figure shall give an overview which timeouts exist and when they are applied.
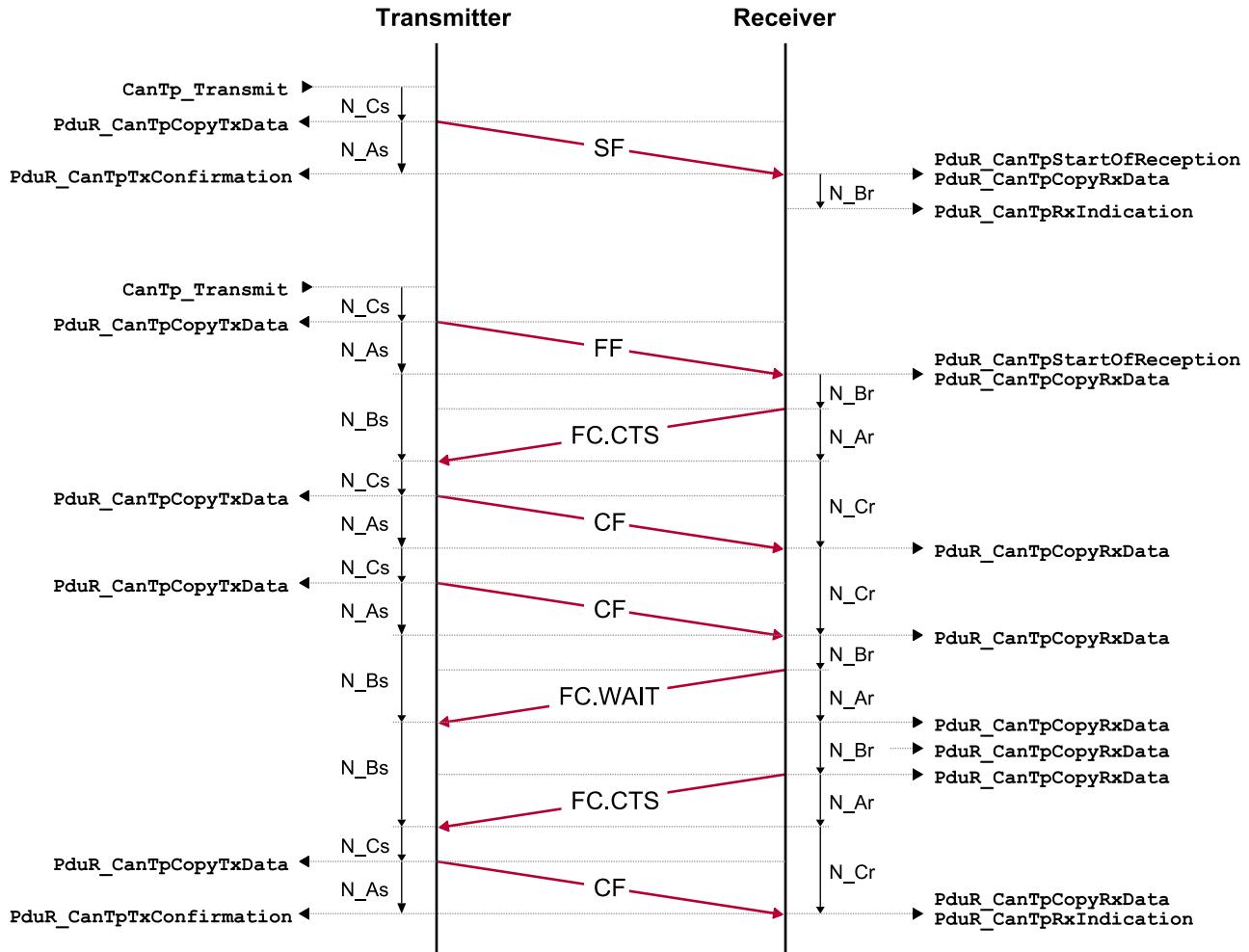


Figure 2-6    Connection Timing

| Timeout | Description |
|---------|-------------|
| Transmission | |
| N_As | Timeout when waiting for the TxConfirmation of a transmitted SF, FF, or CF |
| N_Bs | Timeout when waiting for a Flow Control |
| N_Cs | Time until next CF has to be transmitted, or timeout when waiting a buffer |
| Reception | |
| N_Ar | Timeout when waiting for the TxConfirmation of a transmitted FC |
| N_Br | Time before the transmission of the next FC (see 2.8.2.3) or Timeout when waiting after SF reception for a buffer |
| N_Cr | Timeout when waiting for next CF |

Table 2-8    Connection Timing Parameters

### 2.8.2    Timing Considerations and Jitter

#### 2.8.2.1    Jitter

The time base for all timeout parameters is the main function period of the CanTp. Because the starting point of each timeout may be located sometime between two main function calls, a jitter of up to one task cycle can occur and must be taken into account.

The CanTp tries to compensate this by adding one task cycle to the configured timings, so the observed timings can be longer, but will never be shorter than configured.

#### 2.8.2.2    Separation Time

ISO 15765-2 specifies two timing parameters which determine the time between the transmissions of two consecutive frames.

**STmin** is the minimum separation time, which is provided by the receiver. If the transmitter sends the CFs faster than requested, there is no guarantee that the receiver of the segmented data transmission will correctly receive and process all frames. Another purpose of STmin is to reduce the bus load produced by CanTp communication.

**N_Cs** is the maximum separation time, after which the transmission of the next CF has to be started. If the delay is longer than N_Cs, the receiver side may detect an N_Cr timeout.

> **! Caution**
> In case of a conflict between the configured N_Cs and the requested STmin (N_Cs < STmin), the CanTp will transmit the next CF after the end of N_Cs and therefore violate STmin.

Basically, for STmin the same provisions regarding jitter are made as for all other timings (see 2.8.2.1). However, some STmin specific characteristics apply here:

> in burst mode (STmin = 0) the CFs are sent as fast as possible, i.e. from the context of the TxConfirmaton of the previous CF. The observed separation time then only depends on the bus load.

> when a microsecond STmin is requested (0xF1…0xF9) the CF is always sent on the next task. No additional task cycle is added. Therefore if the bus load is very high and the TxConfirmation is delayed too long, the observed separation time might sporadically be above the given value.

> when STmin supervision is done not internally but by the application (see 2.1.2.11), the CanTp only transmits the CF by itself in case N_Cs expires.

> **FAQ: Why is the observed STmin so much longer than the requested STmin?**
> Because of the jitter, the CanTp always adds one task cycle to the requested STmin in order to guarantee that the time between two CFs is never below STmin. When the task cycle time is higher than STmin, this may lead to unexpected high separation times (see Table 2-9).

| Requested STmin | Observed ST (5ms cycle time) | Observed ST (10ms cyle time) |
|---|---|---|
| 0ms (burst) | as fast as possible | as fast as possible |
| 100µs | < 5ms | < 10ms |
| 1ms | 5…10ms | 10…20ms |
| 5ms | 5…10ms | 10…20ms |
| 6ms | 10…15ms | 10…20ms |
| 10ms | 10…15ms | 10…20ms |
| 11ms | 15…20ms | 20…30ms |

Table 2-9     Examples for requested and observed separation times

### 2.8.2.3     Implementation of N_Br

The ISO 157675-2 defines the parameter N_Br as follows:

  *N_Br: Time until transmission of the next Flow Control N_PDU*

Since N_Br is only a performance parameter and no timeout that must be applied in case of erroneous behavior, it can be seen as the maximum time after which a Flow Control has to be transmitted. Considering this, the following behavior has been implemented in the CanTp:

> After the reception of a First Frame, the first Flow Control is transmitted immediately (N_Br = 0ms).

> After the reception of the last Consecutive Frame in a block, the next Flow Control is transmitted immediately (N_Br = 0ms).

> After FC.WAIT, the subsequent FC.CTS is transmitted as soon as sufficient buffer is provided. If no buffer is provided, the next FC.WAIT is transmitted after N_Br.

> **Note**
> The flow status (CTS, WAIT, OVFL) depends on the result of the buffer provision.
>
> FC.WAIT is sent if the buffer is temporarily unavailable or too small. In this case the receiver will continue to send FC.WAIT until the buffer status changes or the maximum number of wait frames (configuration parameter WFTmax) is reached.

# 3 Integration

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

## 3.1 Embedded Implementation

The delivery of the CanTp contains these source code files:

| File Name | Description | Integration Tasks |
|---|---|---|
| CanTp.c | Source file of the CanTp. | - |
| CanTp.h | Header file of the CanTp. | - |
| CanTp_Cbk.h | Header file with CanTp callback function prototypes. | - |
| CanTp_Types.h | Header file with global CanTp type definitions. | - |
| CanTp_Priv.h | Header file with internal CanTp definitions. | - |
| CanTp_Cfg.h | Generated header file containing pre-compile switches and providing symbolic defines. | - |
| CanTp_Lcfg.h | Generated header file containing link-time configurable symbols. | - |
| CanTp_Lcfg.c | Generated source file containing link-time configurable parameters. | - |
| CanTp_PBcfg.h | Generated header file containing post-build-time configurable parameters. | - |
| CanTp_PBcfg.c | Generated source file containing post-build-time configurable parameters. | - |
| CanTp_Lcfg_<OsApplication>.h | Generated header file containing link-time configurable symbols of the <OsApplication>. | - |
| CanTp_Lcfg_<OsApplication>.c | Generated source file containing link-time configurable parameters of the <OsApplication>. | - |
| CanTp_PBcfg_<OsApplication>.h | Generated header file containing post-build-time configurable parameters of the <OsApplication>. | - |
| CanTp_PBcfg_<OsApplication>.c | Generated source file containing post-build-time configurable parameters of the <OsApplication>. | - |
| CanTp_rules_generated.mak | Generated make file fragment. Can be found in folder: <GenDataFolder> \ Make \ CanTp_rules_generated.mak | - |

Table 3-1    Implementation files

> **Note**
> The only files that need to be included by other modules and software components are `CanTp.h` and `CanTp_Cbk.h`. These files contain all definitions and inclusions required to use the CanTp.

## 3.2 Critical Sections

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 supports only one exclusive area (CANTP_EXCLUSIVE_AREA_0), which can be entered from task and interrupt level. It protects all internal state data against unintended modification due to concurrent access and is entered from the following APIs:

> CanTp_RxIndication

> CanTp_TxConfirmation

> CanTp_Transmit

> CanTp_ChangeParameter

> CanTp_CancelReceive

> CanTp_CancelTransmit

> CanTp_StopSeparationTime

> CanTp_MainFunction

The exclusive area must lock the interrupts from all bus systems which may affect CanTp operation. So while in CAN-only systems, locking the CAN interrupts is sufficent, e.g. in an ECU which does CAN-FlexRay routing, also the FlexRay interrupts have to be locked.

## 3.3 Buffer Configuration

The CanTp is able to work with segmented reception buffers, i.e. usually it is not necessary to provide a buffer with size of the complete message to be received.

Basically it is sufficient to provide a buffer which is large enough to store the payload of all CFs within a block. The CanTp will adjust the block size and flow status parameters in the FC frames according to the currently available buffer to control the reception data flow. A new buffer is requested before the start of a new block, i.e. when an FC.CTS is sent.

However, some configuration options limit the reception control capabilities of the CanTp. This must be taken into account during configuration of the buffer size in an upper layer module (e.g. in the PduR when using high level routing).

### 3.3.1 Constant Block Size

If the CanTp is not allowed to change the block size during reception (see also 2.1.2.4), the buffer size which is reported to the CanTp upon request at the end of a block should also not change after the first FC.CTS has been transmitted. Otherwise the CanTp will transmit an FC.WAIT until enough buffer is available, which will unnecessarily delay the reception.

### 3.3.2   Zero Block Size

If the Block Size is configured as zero, an FC is only allowed after FF reception. If once a FC.CTS is transmitted, the CanTp has no possibility to delay the reception if the upper layer runs out of buffer. The CanTp will try to get more buffer between two CFs while waiting for STmin, but for better robustness it is recommended to use BS = 0 only if a full buffer is available. Otherwise the connection will be terminated with an error if the CanTp is still waiting for a buffer when the next CF is received.

### 3.3.3   Zero WFTmax

A WFTmax value of zero does suppress the transmission of FC.WAIT, i.e. the CanTp is not allowed to delay reception. If nevertheless a situation occurs where an FC.WAIT is needed, the connection will be terminated. Therefore, similar to the case with constant block size, it is important to provide always enough buffer for one complete block or (if block size is not constant) for at least one CF.

### 3.3.4   Zero STmin

With a STmin of zero, each CF is transmitted immediately after the previous CF. This eliminates the possibility of a buffer request between two CFs in case of BS = 0. If such a situation nevertheless occurs, the connection will be erroneously terminated on reception of the next CF.

Therefore it is highly recommended to provide always a full buffer when BS = 0 and STmin = 0 are used.

## 3.4 Multi Partition Support

### 3.4.1 Concept

CanTp supports distributing its functionality onto several OS Partitions (EcucPartitions). The feature is defined through the following cornerpoints:

> The main source of information regarding the EcucPartition a CanTpChannel is assigned to comes from the EcucPartition of the global Pdus referenced by the channels CanTp[Rx|Tx][Fc]NPdus.

> In case of dynamic channel assignment the number of state machines used to operate Rx- and Tx-transfers is adjusted per OS Partition. Instead of the CanTpGeneral container, the new container CanTpEcucPartition contains the dynamic channel assignment parameters CanTpNum[Rx|Tx]Channels. The assignment of this container to a specific OS Partition is done by its CanTpEcucPartitionRef.

> Multi Partition Support is turned on via CanTpGeneral switch CanTpSupportMultiPartition.

> Validators have been added to check the following configuration constraints:

>> The EcucPartition of a CanTpChannel (initially defined by the channels NPdus) is properly mirrored in the CanTpChannel's parameter CanTpChannelEcucPartitionRef.

>> The CanTpChannel and all its NPdus and NSdus shall only reference one and the same EcucPartition.

>> Each EcucPartition referenced by configuration items of the CanTp has to be referenced by an OS Application (OsApplication container), because the short name of the referenced OS Application is imperatively used as name-token in the generated data of the CanTp.

> In the Multi Partition use case, generated data is split into some files that are commonly used by all EcucPartitions and some files that are EcucPartition- (or rather OsApplication-) specific. An AUTOSAR Makefile fragment is generated in order to properly build all generated files. Refer to table Table 3-1 for details on the names of the generated files.

**Note**
The EcucPartition info of CanTp[Rx|Tx][Fc]NPdus is mainly defined by the hardware-related configuration of the CanDrv and the CanIf and validated by validators of those two components. Thus, a change of the EcucPartition of the global Pdu of a CanTp[Rx|Tx][Fc]NPdu is only offered as a hint for a "Manual Action". CanTp will solely offer SolvingActions to change the EcucPartition references of CanTpChannels, its CanTp[Rx|Tx]Sdus and of CanTpEcucPartitions.

### 3.4.2 Performance Considerations

In order to not call OS function GetApplicationID() too often in the Multi Partition use case, during runtime, the following CanTp APIs use a static mapping of their passed [Rx|Tx]N[P|S]duId to the particular OsApplicationId:

> CanTp_Transmit()

> CanTp_CancelTransmit()

> CanTp_CancelReceive()

> CanTp_RxIndication()

> CanTp_TxConfirmation()

The following CanTp APIs use GetApplicationID(), because they are not called very often, during runtime():

> CanTp_Init()

> CanTp_PreInit()

> CanTp_Shutdown()

**Note**
For simplicity, the API CanTp_MainFunction() and its sub-functions CanTp_MainFunction[Rx|Tx]() currently use GetApplicationID().

# 4 API Description

For an interfaces overview please see Figure 1-2.

## 4.1 Services provided by CanTp

### 4.1.1 CanTp_InitMemory

| Prototype | |
|---|---|
| void **CanTp_InitMemory** ( void ) | |
| **Parameter** | |
| N/A | N/A |
| **Return code** | |
| Void | N/A |
| **Functional Description** | |
| Service to initialize module global variables at power up. This function initializes the variables in \*_INIT_\* sections and should be used in case they are not initialized by the startup code. | |
| **Particularities and Limitations** | |
| > This function must be called prior to CanTp_Init<br>> This function can be called from any context.<br>> This function is non-reentrant.<br>> This function is synchronous. | |

Table 4-1    CanTp_InitMemory

## 4.1.2 CanTp_PreInit()

| Prototype | |
|---|---|
| void **CanTp_PreInit** ( const CanTp_ConfigType*  CfgPtr ) | |
| **Parameter** | |
| CfgPtr | Pointer to the CanTp post-build configuration data. |
| | In configurations supporting multiple variants or post-build, a `#define` with a config pointer for each available Configuration Set is generated. |
| | In simple precompile configurations or in the Multi Partition use case, the parameter is not used by the CanTp and can be set to NULL. |
| **Return code** | |
| Void | N/A |
| **Functional Description** | |
| This function pre-initializes the CanTp module. | |
| **Particularities and Limitations** | |
| > This function has to be called by the startup code of the ECU.<br>> This function can be called from any context.<br>> This function is non-reentrant.<br>> This function is synchronous. | |

Table 4-2    CanTp_PreInit()

## 4.1.3 CanTp_Init()

| Prototype | |
|---|---|
| void **CanTp_Init** ( const CanTp_ConfigType*  CfgPtr ) | |
| **Parameter** | |
| CfgPtr | Pointer to the CanTp post-build configuration data. |
| | This parameter is not used by the CanTp and can be set to NULL. |
| **Return code** | |
| Void | N/A |
| **Functional Description** | |
| In the Multi Partition use case this function initializes the EcucPartition specific parts of the CanTp. Otherwise this function initializes the whole CanTp module. | |
| **Particularities and Limitations** | |
| > This function can be called from any context.<br>> This function is non-reentrant.<br>> This function is synchronous. | |

Table 4-3    CanTp_Init()

## 4.1.4 CanTp_Shutdown()

| Prototype | |
|---|---|
| void **CanTp_Shutdown** ( void ) | |
| **Parameter** | |
| N/A | N/A |
| **Return code** | |
| Void | N/A |
| **Functional Description** | |
| This function is called to shut down the CanTp module. | |
| **Particularities and Limitations** | |
| > This function can be called from any context. | |
| > This function is non-reentrant. | |
| > This function is synchronous. | |

Table 4-4     CanTp_Shutdown()

## 4.1.5 CanTp_MainFunction()

| Prototype | |
|---|---|
| void **CanTp_MainFunction** ( void ) | |
| **Parameter** | |
| N/A | N/A |
| **Return code** | |
| Void | N/A |
| **Functional Description** | |
| The main function for scheduling the CanTp. | |
| **Particularities and Limitations** | |
| > This function can be called from any context. | |
| > This function is non-reentrant. | |
| > This function is synchronous. | |

Table 4-5     CanTp_MainFunction()

## 4.1.6    CanTp_MainFunctionRx()

| Prototype | |
|---|---|
| void **CanTp_MainFunctionRx** ( void ) | |
| **Parameter** | |
| N/A | N/A |
| **Return code** | |
| void | N/A |
| **Functional Description** | |
| The main function for scheduling the receive channels of the CanTp. This function is only available if the split MainFunction feature is activated | |
| **Particularities and Limitations** | |

> This function can be called from any context.

> This function is non-reentrant.

> This function is synchronous.

> This API is optional and can be deactivated (see 2.1.2.1)
> Compiler switch: CANTP_RXTX_MAINFUNCTION_API
> Configuration attribute: CanTpEnableSplitMainFunction

Table 4-6    CanTp_MainFunctionRx()

## 4.1.7 CanTp_MainFunctionTx()

| Prototype | |
|---|---|
| void **CanTp_MainFunctionTx** ( void ) | |
| **Parameter** | |
| N/A | N/A |
| **Return code** | |
| void | N/A |
| **Functional Description** | |
| The main function for scheduling the transmit channels of the CanTp. This function is only available if the split MainFunction feature is activated | |
| **Particularities and Limitations** | |
| > This function can be called from any context.<br>> This function is non-reentrant.<br>> This function is synchronous.<br>> This API is optional and can be deactivated (see 2.1.2.1)<br>Compiler switch: CANTP_RXTX_MAINFUNCTION_API<br>Configuration attribute: CanTpEnableSplitMainFunction | |

Table 4-7 CanTp_MainFunctionTx()

## 4.1.8 CanTp_GetVersionInfo()

| Prototype | |
|---|---|
| void **CanTp_GetVersionInfo** ( Std_VersionInfoType* versioninfo ) | |
| **Parameter** | |
| versioninfo | reference to a variable where to store the version information of the CanTp |
| **Return code** | |
| void | N/A |
| **Functional Description** | |
| This function returns the version information of the CanTp module.<br>The version information includes: Module Id, Vendor Id and Vendor specific version numbers.<br>The version numbers are BCD-coded. | |
| **Particularities and Limitations** | |
| > This function can be called from any context.<br>> This function is reentrant.<br>> This function is synchronous.<br>> This API is optional and can be deactivated<br>Compiler switch: CANTP_VERSION_INFO_API<br>Configuration attribute: CanTpVersionInfoApi | |

Table 4-8 CanTp_GetVersionInfo()

## 4.1.9 CanTp_Transmit()

| Prototype | |
|---|---|
| `Std_ReturnType` **`CanTp_Transmit`** `( PduIdType CanTpTxSduId, const PduInfoType* CanTpTxInfoPtr )` | |
| **Parameter** | |
| `CanTpTxSduId` | Unique CanTp identifier of the N-SDU to be transmitted. Range: 0..(maximum number of Tx N-SDU IDs ) - 1 |
| `CanTpTxInfoPtr` | This reference to a PduInfoType structure contains the length to be transmitted (SduLength) and a data pointer (SduDataPointer). Only the SduLength is used by CanTp_Transmit. The data to be transmitted is requested separately by the CanTp. Thereto the function PduR_CanTpCopyTxData is used. |
| **Return code** | |
| `Std_ReturnType` | E_OK: The transmit request has been started successfully E_NOT_OK: The request cannot be started (e.g. a transmit request is in progress with the same N-SDU identifier) |
| **Functional Description** | |
| This service is used to request the transfer of segmented data.<br><br>If data length is less than 7 or 6, depending on the addressing format (standard, extended, mixed11, mixed29 or normal fixed), a SF N-PDU is sent. Otherwise, if data length is greater than 7 or 6, a multiple frame transmission session is initiated.<br><br>When the transmit request has been completed, the CanTp notifies the upper layer by calling the PduR_CanTpTxConfirmation callback function.<br><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. | |
| **Particularities and Limitations** | |
| > This function can be called from any context.<br>> This function is reentrant.<br>> This function can be configured to be synchronous or asynchronous (see 0). | |

Table 4-9     CanTp_Transmit()

## 4.1.10 CanTp_CancelReceive()

| Prototype | |
|---|---|
| `Std_ReturnType` **`CanTp_CancelReceive`** `( PduIdType CanTpRxSduId )` | |
| **Parameter** | |
| `CanTpRxSduId` | Identifier of the Rx N-SDU, for which a reception shall be cancelled. Range: 0..(maximum number of Rx N-SDU IDs) - 1 |
| **Return code** | |
| `Std_ReturnType` | E_OK: Cancellation request of the specified N-SDU is accepted. E_NOT_OK: Cancellation request is rejected; the reason can be that the request is issued for an N-SDU that is not segmented or that is not in the reception process. |
| **Functional Description** | |
| This service is used to cancel the ongoing reception of an N-SDU. | |
| **Particularities and Limitations** | |

> This function can be called from any context.

> This function is non-reentrant.

> This function is synchronous.

> This API is optional and can be deactivated
> Compiler switch: CANTP_RC
> Configuration attribute: CanTpRc

Table 4-10    CanTp_CancelReceive()

## 4.1.11 CanTp_CancelTransmit()

| Prototype | |
|---|---|
| Std_ReturnType **CanTp_CancelTransmit** ( PduIdType CanTpTxSduId ) | |
| **Parameter** | |
| CanTpTxSduId | Identifier of the Tx N-SDU, for which a transmission shall be cancelled.<br>Range: 0..(maximum number of Tx N-SDU IDs) - 1 |
| **Return code** | |
| Std_ReturnType | E_OK: Cancellation request of the specified N-SDU is accepted.<br>E_NOT_OK: Cancellation request is rejected; the reason can be that request is issued for an N-SDU that is not segmented, request is issued after the last CF has been requested for transmission or cancellation is not possible for the related N-SDU due to configuration. |
| **Functional Description** | |
| This service is used to cancel the ongoing transmission of an N-SDU. | |
| **Particularities and Limitations** | |

> This function can be called from any context.
> This function is non-reentrant.
> This function is synchronous.
> This API is optional and can be deactivated
Compiler switch: CANTP_TC
Configuration attribute: CanTpTc

Table 4-11   CanTp_CancelTransmit()

## 4.1.12 CanTp_ChangeParameter()

| Prototype | |
|---|---|
| `Std_ReturnType CanTp_ChangeParameter ( PduIdType id, TPParameterType parameter, uint16 value )` | |
| **Parameter** | |
| `id` | Identifier of the Rx N-SDU, for which a parameter shall be changed. Range: 0..(maximum number of Rx N-SDU IDs) - 1 |
| `parameter` | Parameter type of which the value has to be changes (TP_BS or TP_STMIN). |
| `value` | The new value of the parameter. |
| **Return code** | |
| `Std_ReturnType` | E_OK: request is accepted. E_NOT_OK: request is not accepted. |
| **Functional Description** | |
| This service is used to request the change of reception parameters BS and STmin for a specified N-SDU. Modification of parameters is only allowed if currently no reception for the respective N-SDU is in progress. | |
| **Particularities and Limitations** | |

> This function can be called from any context.

> This function is non-reentrant.

> This function is synchronous.

> This API is optional and can be deactivated
  Compiler switch: CANTP_ENABLE_CHANGE_PARAM
  Configuration attribute: CanTpChangeParameterApi

Table 4-12   CanTp_ChangeParameter()

## 4.1.13 CanTp_ReadParameter()

| Prototype | |
|---|---|
| `Std_ReturnType` **`CanTp_ReadParameter`** `( PduIdType id, TPParameterType parameter, uint16* value )` | |
| **Parameter** | |
| `id` | Identifier of the Rx N-SDU, for which a flow control parameter shall be read. Range: 0..(maximum number of Rx N-SDU IDs) - 1 |
| `parameter` | Parameter type of which the value has to be read (TP_BS or TP_STMIN). |
| `value` | Pointer where the parameter value will be stored. |
| **Return code** | |
| `Std_ReturnType` | E_OK: request is accepted. E_NOT_OK: request is not accepted. |
| **Functional Description** | |
| This service is used to read the current value of reception parameters BS and STmin for a specified N-SDU. | |
| **Particularities and Limitations** | |
| > This function can be called from any context. > This function is non-reentrant. > This function is synchronous. > This API is optional and can be deactivated Compiler switch: CANTP_ENABLE_READ_PARAM Configuration attribute: CanTpReadParameterApi | |

Table 4-13    CanTp_ReadParameter()

## 4.2 Services used by CanTp

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.

| Component | API |
|---|---|
| CanIf | CanIf_Transmit<br>CanIf_CancelTransmit |
| Dcm | Dcm_OnRequestDetection |
| Det | Det_ReportError |
| EcuM | EcuM_BswErrorHook |
| PduR | PduR_CanTpCopyRxData<br>PduR_CanTpCopyTxData<br>PduR_CanTpRxIndication<br>PduR_CanTpStartOfReception<br>PduR_CanTpTxConfirmation<br>PduR_CanTpChangeParameterConfirmation<br>PduR_CanTpTransmit (if the PduR is a lower layer)<br>PduR_CanTpCancelTransmit (if the PduR is a lower layer) |
| SchM | SchM_Enter_CanTp_*<br>SchM_Exit_CanTp_* |
| Os | Os_GetApplicationId |

Table 4-14    Services used by the CanTp

## 4.3 Callback Functions

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.

### 4.3.1 CanTp_RxIndication()

| Prototype | |
|---|---|
| void **CanTp_RxIndication** ( PduIdType CanTpRxPduId, const PduInfoType* CanTpRxPduPtr ) | |
| **Parameter** | |
| CanTpRxPduId | Identifier of the Rx N-PDU that have been received. Range: 0..(maximum number of Rx N-PDU IDs) - 1 |
| CanTpRxPduPtr | Reference to structure with the size of the received N-PDU (SduLength) and to the payload data (SduDataPointer) |
| **Return code** | |
| void | N/A |
| **Functional Description** | |
| This function is called by the CAN Interface after a successful reception of an N-PDU. | |
| **Particularities and Limitations** | |
| > This function can be called from any context. > This function is reentrant. > This function is synchronous. | |

Table 4-15   CanTp_RxIndication()

### 4.3.2 CanTp_TxConfirmation()

| Prototype | |
|---|---|
| void **CanTp_TxConfirmation** ( PduIdType TxPduId ) | |
| **Parameter** | |
| TxPduId | Identifier of the Tx N-PDU that have been transmitted successfully. Range: 0..(maximum number of Tx N-PDU IDs) - 1 |
| **Return code** | |
| void | N/A |
| **Functional Description** | |
| This function is called by the CAN Interface to confirm the successful transmission of an N-PDU. | |
| **Particularities and Limitations** | |
| > This function can be called from any context. > This function is reentrant. > This function is synchronous. | |

Table 4-16   CanTp_TxConfirmation()

### 4.3.3 CanTp_StopSeparationTime()

| Prototype | |
|---|---|
| void **CanTp_StopSeparationTime**(PduIdType CanTpTxSduId) | |
| **Parameter** | |
| CanTpTxSduId | Symbolic name value of the Tx connection<br><br>Note: this is the same ID as it has been passed to the application in the StartSeparationTime call-out. |
| **Return code** | |
| – | - |
| **Functional Description** | |
| Called by the application to trigger transmission of the next CF when its separation timer expired. | |
| **Particularities and Limitations** | |
| > Feature „STmin by Application" must be active (see 2.1.2.11)<br><br>> The request to do the separation time handling must have been accepted by the application (see 4.4.1). If CanTp_StopSeparationTime() is called before Appl_StartSeparationTime returns (e.g. in case of a fast timer interrupt), the CanTp assumes a positive result and accepts the function call.<br><br>> In the context of this function, the CanTp will request the CF payload from its upper layer and transmit the next CF | |

Table 4-17    CanTp_StopSeparationTime()

## 4.4 Configurable Interfaces

### 4.4.1 Appl_StartSeparationTime()

| Prototype | |
|---|---|
| boolean **Appl_StartSeparationTime**(PduIdType CanTpTxSduId, uint8 STmin) | |
| **Parameter** | |
| CanTpTxSduId | Symbolic name value of the Tx connection. |
| STmin | STmin value from the flow control (encoding according to ISO). |
| **Return code** | |
| TRUE | request to handle Separation Time is accepted by the application |
| FALSE | request to handle Separation Time is rejected and will be done by CanTp |
| **Functional Description** | |
| Called by the CanTp to notify the application that a separation timer need to be started. | |
| **Particularities and Limitations** | |
| > The function is called from the TxConfirmation context<br><br>> The actual name of the function is configured by the parameter 'CanTp/CanTpGeneral/CanTpApplSTminStartFunction' | |

Table 4-18    Appl_StartSeparationTime()

### 4.4.2 Notification Functions

Additional notification callouts can be defined to notify the application about the reception or transmission of CanTp frames. This may be used for project specific extensions or workarounds. The function declarations are provided in `CanTp_Cbk.h`. To activate a callout, an according compiler switch must be defined to STD_ON in a user config file.

#### 4.4.2.1 Appl_CanTpRxSFIndication()

| Prototype | |
|---|---|
| `void Appl_CanTpRxSFIndication (PduIdType      PduRRxPduId,` <br> `                            const PduInfoType* PduInfoPtr);` | |
| **Parameter** | |
| PduRRxPduId | PduId of the connection, which is defined by the PduR; the same Id is passed to PduR in PduR_CanTpRxIndication(). |
| PduInfoPtr | Reference to structure with the size (SduLength) and the CAN frame content (SduDataPointer) of the single frame. |
| **Return code** | |
| – | - |
| **Functional Description** | |
| Function is called upon successful reception of a single frame N-PDU before the call of PduR_CanTpStartOfReception(). | |
| **Particularities and Limitations** | |
| > To activate the callout, CANTP_APPL_RX_SF_INDICATION must be defined to STD_ON | |

Table 4-19    Appl_CanTpRxSFIndication()

#### 4.4.2.2 Appl_CanTpRxFFIndication()

| Prototype | |
|---|---|
| `void Appl_CanTpRxFFIndication (PduIdType      PduRRxPduId,` <br> `                            const PduInfoType* PduInfoPtr);` | |
| **Parameter** | |
| PduRRxPduId | PduId of the connection, which is defined by the PduR; the same Id is passed to PduR in PduR_CanTpRxIndication(). |
| PduInfoPtr | Reference to structure with the size (SduLength) and the CAN frame content (SduDataPointer) of the first frame. |
| **Return code** | |
| – | - |
| **Functional Description** | |
| Function is called upon successful reception of a first frame N-PDU before the call of PduR_CanTpStartOfReception(). | |
| **Particularities and Limitations** | |
| > To activate the callout, CANTP_APPL_RX_FF_INDICATION must be defined to STD_ON | |

Table 4-20    Appl_CanTpRxFFIndication()

### 4.4.2.3 Appl_CanTpRxCFIndication()

| Prototype | |
|---|---|
| void **Appl_CanTpRxCFIndication** (PduIdType         PduRRxPduId,<br>                        const PduInfoType* PduInfoPtr); | |
| **Parameter** | |
| PduRRxPduId | PduId of the connection, which is defined by the PduR; the same Id is passed to PduR in PduR_CanTpRxIndication(). |
| PduInfoPtr | Reference to structure with the size (SduLength) and the CAN frame content (SduDataPointer) of the consecutive frame. |
| **Return code** | |
| – | - |
| **Functional Description** | |
| Function is called upon successful reception of a consecutive frame N-PDU before the call of PduR_CanTpCopyRxData(). | |
| **Particularities and Limitations** | |
| > To activate the callout, CANTP_APPL_RX_CF_INDICATION must be defined to STD_ON | |

Table 4-21    Appl_CanTpRxCFIndication()

### 4.4.2.4 Appl_CanTpFrameTransmission()

| Prototype | |
|---|---|
| void **Appl_CanTFrameTransmission** (PduIdType         CanIfTxPduId,<br>                       const PduInfoType* PduInfoPtr); | |
| **Parameter** | |
| CanIfTxPduId | PduId of the transmitted CAN message, which is defined by the CanIf; the same Id is passed to CanIf_Transmit(). |
| PduInfoPtr | Reference to structure with the size (SduLength) and the content (SduDataPointer) of the transmitted CAN frame. |
| **Return code** | |
| – | - |
| **Functional Description** | |
| Function is called if transmission of a CanTp N-PDU has successfully been started (CanIf_Transmit() has been called and returned E_OK). | |
| **Particularities and Limitations** | |
| > To activate the callout, CANTP_APPL_FRAME_TRANSMISSION must be defined to STD_ON | |

Table 4-22    Appl_CanTpFrameTransmission()

### 4.4.2.5   Appl_CanTpFrameTxConfirmation()

| Prototype | |
|---|---|
| void **Appl_CanTFrameTxConfirmation** (PduIdType CanIfTxPduId); | |
| **Parameter** | |
| CanIfTxPduId | PduId of the transmitted CAN message, which is defined by the CanIf; the same Id is passed to CanIf_Transmit(). |
| **Return code** | |
| – | - |
| **Functional Description** | |
| Function is called if a CanTp N-PDU has successfully been transmitted (at the beginning of CanTp_TxConfirmation). | |
| **Particularities and Limitations** | |
| > To activate the callout, CANTP_APPL_FRAME_CONFIRMATION must be defined to STD_ON | |

Table 4-23   Appl_CanTpFrameTransmission()

### 4.4.2.6   Appl_CanTpSduRxIndication()

| Prototype | |
|---|---|
| void **Appl_CanTpSduRxIndication** (PduIdType PduRRxSduId, CanTp_NotificationType NotifyCode); | |
| **Parameter** | |
| PduRRxSduId | PduId of the received CANTP N-SDU. The same Id is used for calls to PduR_CanTpRxIndication(). |
| NotifyCode | Result of the TP reception. |
| **Return code** | |
| – | - |
| **Functional Description** | |
| Function is called if a CANTP N-SDU has successfully been received or to report an error that occurred during the reception. | |
| **Particularities and Limitations** | |
| > To activate the callout, CANTP_APPL_RX_SDU_INDICATION must be defined to STD_ON | |

Table 4-24   Appl_CanTpSduRxIndication()

## 4.4.2.7 Appl_CanTpSduTxConfirmation()

| Prototype | |
|---|---|
| void **Appl_CanTpSduTxConfirmation** (PduIdType PduRTxSduId, CanTp_NotificationType NotifyCode); | |
| **Parameter** | |
| PduRTxSduId | PduId of the transmitted CANTP N-SDU. The same Id is used for calls to PduR_CanTpTxConfirmation(). |
| NotifyCode | Result of the TP transmission. |
| **Return code** | |
| – | - |
| **Functional Description** | |
| Function is called if a CANTP N-SDU has successfully been transmitted or to report an error that occurred during the transmission. | |
| **Particularities and Limitations** | |
| > To activate the callout, CANTP_APPL_TX_SDU_CONFIRMATION must be defined to STD_ON | |

Table 4-25   Appl_CanTpSduTxConfirmation()

### 4.4.3 Custom addressing callout functions

The following functions are required by the CanTp when using custom addressing.

### 4.4.3.1 Appl_CanTpRxIndicationCallout()

| Prototype | |
|---|---|
| `Std_ReturnType` **`Appl_CanTpRxIndicationCallout`** `(`<br>`                           const PduInfoType* NPduPayloadPtr,`<br>`                           const PduInfoType* NPduMetadataPtr,`<br>`                           PduInfoType* ResMetadataOfNSduOutPtr,`<br>`                           uint8 * ResMetadataOfFcNPduOutPtr,`<br>`                           PduInfoType* ResPayloadOfFcNPduInOutPtr)` | |
| **Parameter** | |
| `NPduPayloadPtr` | Payload as provided by the CanIf. |
| `NPduMetadataPtr` | N-PDU metadata as provided by the CanIf. |
| `ResMetadataOfNSduOutPtr` | Resulting N-SDU metadata that identifies an Rx connection and could be passed to the upper layer when calling PduR_CanTpStartOfReception.<br><br>The function is called with the expected length already set (`ResMetadataOfNSduOutPtr->SduLength`). This length is the same as the metadata length configured for the N-SDU. |
| `ResMetadataOfFcNPduOutPtr` | Resulting FC N-PDU metadata that could be passed to the CanIf when sending the FlowControl N-PDUs of the Rx connection identified by the resulting N-SDU metadata.<br><br>This metadata influences the identifier of the CAN frames used to send the FlowControl N-PDUs. Its length shall be the same as the metadata length configured for the data and FC N-PDUs (`NPduMetadataPtr->SduLength`). |
| `ResPayloadOfFcNPduInOutPtr` | Not used. |
| **Return code** | |
| `E_OK` | Metadata interpretation succeeded. Continue processing the received N-PDU. |
| `E_NOT_OK` | Metadata interpretation failed. Discard the received N-PDU. |
| **Functional Description** | |
| This function is called by the CanTp each time the metadata of an Rx N-SDU (configured with custom addressing) has to be determined based on the metadata of a received N-PDU.<br><br>The returned metadata is used by the CanTp to identify the data N-PDUs belonging to an Rx connection and to send flow control N-PDUs with the correct CAN ID. | |

| Particularities and Limitations |
| --- |

> The function is called from the CanTp_RxIndication context.

> The actual name of the function is configured with the parameter
CanTp/CanTpConfig/CanTpChannel/CanTpRxNSdu/CanTpCustomAddressingCallout

> This function is available only if custom addressing is enabled.

**Caution**
The callout implementation shall satisfy the following points:

1. `ResMetadataOfNSduOutPtr->SduLength` shall not be modified.

2. Exactly `ResMetadataOfNSduOutPtr->SduLength` bytes shall be copied to `ResMetadataOfNSduOutPtr->SduDataPtr`.

3. Exactly `NPduMetadataPtr->SduLength` bytes shall be copied to `ResMetadataOfFcNPduOutPtr`.

4. `ResPayloadOfFcNPduInOutPtr` shall not be used.

Table 4-26    Appl_CanTpRxIndicationCallout()

### 4.4.3.2 Appl_CanTpTransmitCallout()

| Prototype | |
|---|---|
| `Std_ReturnType` **`Appl_CanTpTransmitCallout`** `(const PduInfoType* MetadataOfNSduPtr,`<br>`                                PduInfoType* ResMetadataOfDataNPduOutPtr,`<br>`                                uint8 * ResMetaDataOfFcNPduOutPtr,`<br>`                                PduInfoType* ResPayloadOfDataNPduInOutPtr,`<br>`                                uint8 * ResPayloadOfFcNPduOutPtr)` | |
| **Parameter** | |
| `MetadataOfNSduPtr` | N-SDU metadata as provided by the upper layer. Identifies a Tx connection. |
| `ResMetadataOfDataNPduOutPtr` | Resulting data N-PDU metadata to be passed to the CanIf when sending the data N-PDUs of the Tx connection identified by the passed N-SDU metadata.<br><br>This metadata influences the identifier of the CAN frames used to send the data N-PDUs.<br><br>The function is called with the expected length already set (`ResMetadataOfDataNPduOutPtr->SduLength`). This length is the same as the metadata length configured for the data and FC N-PDUs. |
| `ResMetaDataOfFcNPduOutPtr` | Resulting FC N-PDU metadata to be compared against the metadata of received FC N-PDUs. FC N-PDUs with matching metadata are considered to belong to the Tx connection identified by the passed N-SDU metadata.<br><br>Its length shall be the same as the metadata length configured for the data and FC N-PDUs (`ResMetadataOfDataNPduOutPtr->SduLength`). |
| `ResPayloadOfDataNPduInOutPtr` | Not used. |
| `ResPayloadOfFcNPduOutPtr` | Not used. |
| **Return code** | |
| `E_OK` | Metadata interpretation succeeded. Start a Tx connection. |
| `E_NOT_OK` | Metadata interpretation failed. Don't start a Tx connection (CanTp_Transmit will return E_NOT_OK) |
| **Functional Description** | |
| This function is called by the CanTp each time the metadata of a Tx N-SDU (configured with custom addressing) has to be interpreted.<br><br>The returned N-PDU metadata is used by the CanTp to identify the FC N-PDUs belonging to a Tx connection and to send data N-PDUs with the correct CAN ID. | |

## Particularities and Limitations

> The function is called from the CanTp_Transmit context.

> The actual name of the function is configured with the parameter CanTp/CanTpConfig/CanTpChannel/CanTpTxNSdu/CanTpCustomAddressingCallout

> This function is available only if custom addressing is enabled.

> **Caution**
> The callout implementation shall satisfy the following points:
> 1. `ResMetadataOfDataNPduOutPtr->SduLength` shall not be modified.
> 2. Exactly `ResMetadataOfDataNPduOutPtr->SduLength` bytes shall be copied to `ResMetadataOfDataNPduOutPtr->SduDataPtr`.
> 3. Exactly `ResMetadataOfDataNPduOutPtr->SduLength` bytes shall be copied to `ResMetaDataOfFcNPduOutPtr`.
> 4. `ResPayloadOfDataNPduInOutPtr` shall not be used.
> 5. `ResPayloadOfFcNPduOutPtr` shall not be used.

Table 4-27    Appl_CanTpTransmitCallout()

# 5 Configuration

The CanTp attributes can be configured with the following tool:

> Configuration in DaVinci Configurator Pro 5

## 5.1 Configuration Variants

The CanTp supports the configuration variant

> `VARIANT-PRE-COMPILE`

> `VARIANT-POST-BUILD-LOADABLE`

> `VARIANT-POST-BUILD-SELECTABLE`

The configuration classes of the CanTp parameters depend on the supported configuration variants. For their definitions please see the CanTp_bswmd.arxml file.

## 5.2 Configuration of Post-Build

The configuration of post-build loadable is described in [8].

In the CanTp, the following configuration changes are possible at post-build time:

> Modify timing parameters of existing N-SDUs

> Activate or disable N-PDU padding of existing N-SDUs (padding must have been globally enabled at pre-compile time)

> Modify flow control parameters of existing Rx N-SDUs (STmin, Block Size, WFTmax)

> Change number of channels (dynamic channel assignment must have been enabled at pre-compile time, see 2.1.2.5)

> Add new N-SDUs

Existing references to N-PDUs can't be changed at post-build time. However, when adding new N-SDUs, N-PDUs configured at pre-compile time can be referenced.

## 5.3 Additional Configuration Hints

### 5.3.1 CanIf Tx Buffering

The CanTp does not implement a retry mechanism in case the CanIf is not able to transmit a TP message. If a call to CanIf_Transmit failed, the connection is terminated.

To avoid unpredictable interruption of active CanTp connections, the Tx buffering feature must be enabled in the CanIf for systems where high bus load and failed transmissions are expected.

### 5.3.2 ISO Performance Requirements

ISO15765-2 defines performance requirements for N_Br and N_Cs (see also 2.8.1), which frequently leads to confusion. The reason for this is that there is usually no clear distinction between **configured timeouts** and the **actual timing** observed on the bus.

For example, a typical OEM specification may look like that:

| Timing Parameter | Value |
|---|---|
| N_As | 1000ms |
| N_Ar | 1000ms |
| N_Bs | 1000ms |
| N_Br | (N_Br + N_Ar) < (0,9*N_Bs) |
| N_Cs | (N_Cs + N_As) < (0,9*N_Cr) |
| N_Cr | 1000ms |

Table 5-1     Example for typical timing parameter specification

Interpreting all timings as configurable parameters would mean, that only a negative value can fulfill the specified equation for N_Br and N_Cs.

However, the intent of the requirement is not to specify a concrete value, but to provide a test criterion for a black box test where e.g. N_Br and N_Ar can't be measured separately. So it should be interpreted as follows:

(actual measured time for N_Br + N_Ar) < (0,9 * configured N_Bs timeout)

**FAQ: Which values should be used for the config parameters N_Cs and N_Br?**

The CanTp tries to send out frames as fast as possible. N_Cs and N_Br are mainly used to abort/complete an operation before it is obvious that the performance time can no longer be fulfilled (see also 2.8.2.2 for N_Cs and 2.8.2.3 for N_Br).

So assuming a typical bus delay far below 100ms, for the example above a good choice for N_Cs and N_Br would be around 800ms.

### 5.3.3 PduR as Lower Layer

The N-SDUs grouped under a channel can be configured to use the PduR as lower layer by following these steps:

1. Set the parameter CanTpSupportedLowerLayers either to `LOWER_LAYER_PDUR` or `LOWER_LAYER_CANIF_AND_PDUR`.

2. Set the parameter CanTpChannelLowerLayer of the channels grouping the relevant N-SDUs to `LOWER_LAYER_PDUR`.

3. Make sure the following parameters are enabled for the CanTp in the PduR:

   > PduRUpperModule and PduRLowerModule

   > PduRCommunicationInterface and PduRTransportProtocol

   > PduRUseTag

   > PduRCancelTransmit (if required)

4. After setting these parameters, create any references missing between the CanTp, the EcuC and the PduR by following the validation messages shown in the DaVinci Configurator Pro 5.

> **Caution**
> The lower layers that will be available for N-SDUs added during post-build time are determined at pre-compile time by the parameter CanTpSupportedLowerLayers.

### 5.3.4 Generic Connections with metadata

A Tx N-SDU is identified as a generic connection as soon as the target of one of the following references is configured with metadata:

> CanTpTxNSdu/CanTpTxNSduRef

> CanTpTxNSdu/CanTpTxNPdu/CanTpTxNPduRef

> CanTpTxNSdu/CanTpRxFcNPdu/CanTpRxFcNPduRef

An Rx N-SDU is identified as a generic connection as soon as the target of one of the following references is configured with metadata:

> CanTpRxNSdu/CanTpRxNSduRef

> CanTpRxNSdu/CanTpRxNPdu/CanTpRxNPduRef

> CanTpRxNSdu/CanTpTxFcNPdu/CanTpTxFcNPduRef

SDUs identified as generic connections are validated depending on their addressing mode, to ensure that the correct metadata length is configured in the related global PDUs and that containers with unnecessary addressing information are not present.

Generic connections are allowed to "share" N-PDUs. For example, several Rx N-SDUs could be configured with the same CanTpRxNSdu/CanTpRxNPdu/CanTpRxNPduRef. If the target N-PDU represents a range of CAN IDs and is configured with metadata, then simultaneous receptions (over different CAN IDs within the range) could take place. The number of simultaneous receptions will be limited by the number of Rx N-SDUs "sharing" the N-PDU.

### 5.3.5 Custom Addressing

For Rx or Tx SDUs with Custom Addressing, the metadata is completely controlled by 2 callout functions to the upper layer. This enables the upper layer to implement an addressing scheme, independent of the configuration.

Custom Addressing for a Tx connection requires a Tx N-SDU to have its:

> CanTpTxNSdu/CanTpTxAddressingFormat set to CANTP_CUSTOM

and the target of all of the following references to be configured with metadata:

> CanTpTxNSdu/CanTpTxNSduRef

> CanTpTxNSdu/CanTpTxNPdu/CanTpTxNPduRef

> CanTpTxNSdu/CanTpRxFcNPdu/CanTpRxFcNPduRef

Custom Addressing for an Rx connection requires an Rx N-SDU to have its:

> CanTpRxNSdu/CanTpRxAddressingFormat set to CANTP_CUSTOM

and the target of all of the following references to be configured with metadata:

> CanTpRxNSdu/CanTpRxNSduRef

> CanTpRxNSdu/CanTpRxNPdu/CanTpRxNPduRef

> CanTpRxNSdu/CanTpTxFcNPdu/CanTpTxFcNPduRef

SDUs identified to have custom addressing are validated to have configured these items:

> CanTpRxNSdu/CanTpCustomAddressingCallout

> CanTpTxNSdu/CanTpCustomAddressingCallout

In addition, the header files that declare the 2 callout functions, listed above, have to be configured in:

> CanTpGeneral/CanTpCustomAddressingCalloutHeaderFile

# 6 Glossary and Abbreviations

## 6.1 Glossary

| Term | Description |
|------|-------------|
| 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. |
| 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. |
| DaVinci Configurator Pro 5 | Configuration and code generation tool for MICROSAR components |
| 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 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. |
| 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 6-1        Glossary

## 6.2    Abbreviations

| Abbreviation | Description |
|---|---|
| AE | Address Extension |
| API | Application Programming Interface |
| AR | AUTOSAR |
| AUTOSAR | Automotive Open System Architecture |
| BS | Block Size |
| BSW | Basis Software |
| BSWMD | Basic Software Module Description |
| CAN | Controller Area Network protocol |
| CAN-FD | CAN with Flexible Data Rate |
| 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 |
| FC | Flow Control |
| FF | First Frame |
| FS | Flow Status Control |
| HIS | Hersteller Initiative Software |
| ID | Identifier |
| ISO | International Standardization Organization |
| LFF | Long First Frame (extended first frame with escape sequence) |
| LSF | Long Single Frame (extended single frame with escape sequence) |
| MICROSAR | Microcontroller Open System Architecture (the Vector AUTOSAR solution) |
| N | Network Layer (used as prefix; e.g. N-PDU or N-SDU) |
| OBD | Undefined |
| OEM | Original Equipment Manufacturer |
| OS | Operating System |
| OVFLW | Overflow |
| PB | Post-Build |
| PDU | Protocol Data Unit |

| | |
|---|---|
| PDUR | PDU Router |
| ROM | Read-Only Memory |
| RFC | Request For Comment |
| RTE | Runtime Environment |
| SA | Source Address |
| SDU | Service Data Unit |
| SF | Single Frame |
| SN | Sequence Number |
| SRS | Software Requirement Specification |
| ST | Separation Time |
| SWC | Software Component |
| SWS | Software Specification |
| TA | Target Address |
| TP | Transport Protocol |
| TPCI | Transport Protocol Control Information |
| WT | Wait |

Table 6-2    Abbreviations

# 7   Contact

Visit our website for more information on

> News

> Products

> Demo software

> Support

> Training data

> Addresses

www.vector.com