Document Title	Specification of TTCAN Interface
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	433

Document Status	Final
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	4.3.1

Document Change History

Date	Release	Changed by	Description
2017-12-08	4.3.1	AUTOSAR Release Management	• Replace Can_ReturnType with Std_ReturnType overlay • Editorial changes
2016-11-30	4.3.0	AUTOSAR Release Management	• Remove CCMSM • Dem API update • Editorial changes
2015-07-31	4.2.2	AUTOSAR Release Management	• Fixed error section • Editorial changes
2014-10-31	4.2.1	AUTOSAR Release Management	• Improved extended production error description • Updated disclaimer • Editorial changes
2014-03-31	4.1.3	AUTOSAR Release Management	• Adapted description of exported TTCAN EcuC containers • Editorial changes
2013-10-31	4.1.2	AUTOSAR Release Management	• Editorial changes
2013-03-15	4.1.1	AUTOSAR Administration	• Updated scope of parameters • Formal update for traceability analysis • Aligned to General Documents • Adapted Production Error Specification

2011-12-22	4.0.3	AUTOSAR Administration	• Updated <User_TriggerTransmit> function with generated artifact from ComStack harmonization • Described behaviour of negative return value of <User_TriggerTransmit>
2010-02-02	3.1.4	AUTOSAR Administration	• Initial Release

Disclaimer

This work (specification and/or software implementation) and the material contained in it, as released by AUTOSAR, is for the purpose of information only. AUTOSAR and the companies that have contributed to it shall not be liable for any use of the work.

The material contained in this work is protected by copyright and other types of intellectual property rights. The commercial exploitation of the material contained in this work requires a license to such intellectual property rights.

This work may be utilized or reproduced without any modification, in any form or by any means, for informational purposes only. For any other purpose, no part of the work may be utilized or reproduced, in any form or by any means, without permission in writing from the publisher.

The work has been developed for automotive applications only. It has neither been developed, nor tested for non-automotive applications.

The word AUTOSAR and the AUTOSAR logo are registered trademarks.

Table of Contents

1 Introduction and functional overview

This specification describes the functionality, API and the configuration for the AUTOSAR Basic Software module TTCAN Interface (called "'TtcanIf"' in this document).

The base for this document is [1, ISO 11898-4]. It is assumed that the reader is familiar with this specification. This document will not describe TTCAN functionality again.

`TtcanIf` is located in the communication hardware abstraction under the communication service layers (i.e. TTCAN State Manager, TTCAN Network Management, TTCAN Transport Protocol, PDU Router). It represents the interface to the services of the `TTCAN Driver` for the upper communication layers.

`TtcanIf` is an extension of the [2, CAN Interface module (CanIf)] so this document shall only provide information and specifications which differ from `CanIf`.

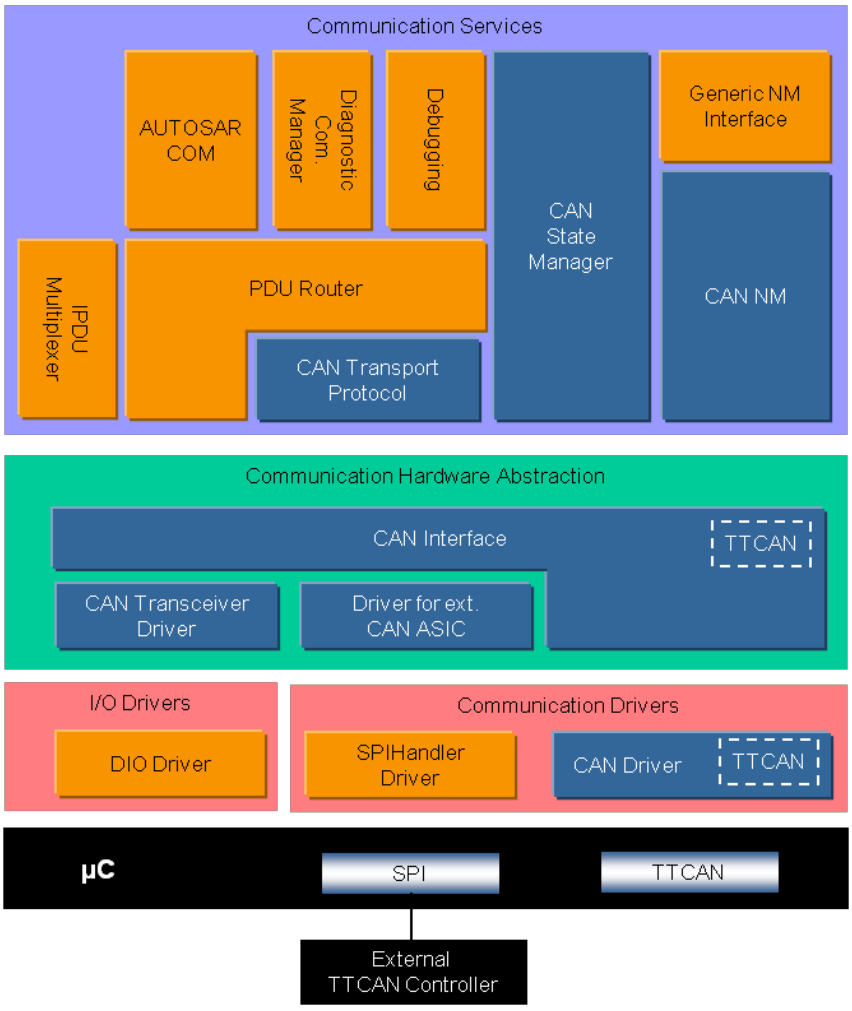

Figure 1.1: AUTOSAR TTCAN Layer Model (see [3])

Messages, which are configured for Exclusive Time Windows, will be transmitted periodically with every Tx_Trigger configured for this message (Continuous Transmission).

Messages, which are configured for Arbitrating Time Windows, will be transmitted only once per Transmit Request (Single Shot).

TtcanIf consists of all TTCAN hardware independent tasks, which belong to the TTCAN communication device drivers of the corresponding ECU. This functionality is implemented once in TtcanIf, so that underlying TTCAN device drivers only focus on access and control of the corresponding specific TTCAN hardware device.

TtcanIf fulfils main control flow and data flow requirements of the PDU Router and upper layer communication modules of the AUTOSAR COM stack: transmit request processing, transmit confirmation / receive indication / error notification and start / stop of a TTCAN Controller and thus waking up / participating on a network. Its data processing and notification API is based on CAN L-PDUs, whereas the APIs for control and mode handling provide a TTCAN Controller related view.

In case of transmit requests TtcanIf completes the L-PDU transmission with corresponding parameters and relays the CAN L-PDU via the appropriate TTCAN Driver to the TTCAN Controller. At reception TtcanIf distributes the received L-PDUs to the upper layer. The assignment between receive L-PDU and upper layer is statically configured. At transmit confirmation TtcanIf is responsible for the notification of upper layers about successful transmission.

TtcanIf provides TTCAN communication abstracted access to the lower layer services for control and supervision of the TTCAN network. TtcanIf forwards the status change requests from the CAN State Manager downwards to the lower layer TTCAN device drivers, and upwards the lower layer events are forwarded by TtcanIf to e.g. the corresponding NM module.

2 Acronyms and Abbreviations

The glossary below includes acronyms and abbreviations relevant to TtcanIf that are not included in the [4, AUTOSAR glossary].

Abbreviation / Acronym:	Description:
"'at system configuration time"'	static configuration parameters stored in TtcanIf; may be defined after compilation of the code of TtcanIf, but have to be defined before the first execution of TtcanIf code.
Arbitrating Time Window	See [1, ISO 11898-4]
Basic Cycle	See [1, ISO 11898-4]
BSW	Basic Software
CanIf	CAN Interface
Communication Job	A TTCAN Communication Job defines the specific communication operation and the assigned execution time.
Continuous Transmission	Contrary to Single Shot a message will be transmitted cyclically even without a new transmit request.
Controller	A (TTCAN-)Controller is a CPU on-chip or external standalone hardware device. One Controller is connected to one physical channel.
Cycle Time	See [1, ISO 11898-4]
Dem	Diagnostic Event Manager
DLC	Data Length Code (part of L-PDU that describes the SDU length)
DLL	Data Link Layer
EcuM	ECU Manager
Exclusive Time Window	See [1, ISO 11898-4]
Gap	See [1, ISO 11898-4]
Global Time	See [1, ISO 11898-4]
Hardware Object	A CAN hardware object is defined as a PDU buffer inside the CAN RAM of the CAN hardware unit / CAN Controller.
ISR	Interrupt Service Routine
JLEF	(TTCAN) Job List Execution Function
Job List	A TTCAN Job List is a list of (maybe different) Communication Jobs sorted according to their respective execution start time.
L-PDU	Protocol Data Unit for the Data Link Layer (DLL)
Local Time	See [1, ISO 11898-4]
Matrix Cycle	See [1, ISO 11898-4]
MCAL	Microcontroller Abstraction Layer
NTU	See [1, ISO 11898-4]
OS	(AUTOSAR) Operating System
PduR	PDU Router
Reference Message	See [1, ISO 11898-4]
SDU	Service Data Unit
Single Shot	A message will be transmitted only once contrary to Continuous Transmission.
System Matrix	See [1, ISO 11898-4]
Time Gap	See [1, ISO 11898-4]
Time Master	See [1, ISO 11898-4]
Time Window	See [1, ISO 11898-4]
Transmission Column	See [1, ISO 11898-4]
TtcanDrv	CAN Driver module with enabled TTCAN functionality
TtcanIf	CAN Interface module with enabled TTCAN functionality
CanNm	CAN Network Management

CanSM	CAN State Manager
CanTp	CAN Transport Protocol
TX	Transmission or transmit
Tx_Trigger	See [1, ISO 11898-4]
UL	Upper layer

3 Related documentation

All documents of the referenced CAN Interface document [2] are also valid for this document.

3.1 Input documents & related standards and norms

Bibliography

[1] ISO 11898-4:2004 - Road vehicles - Controller area network (CAN) - Part 4: Time-triggered communication

[2] Specification of CAN Interface
AUTOSAR_SWS_CANInterface

[3] Layered Software Architecture
AUTOSAR_EXP_LayeredSoftwareArchitecture

[4] Glossary
AUTOSAR_TR_Glossary

[5] General Specification of Basic Software Modules
AUTOSAR_SWS_BSWGeneral

3.2 Related specification

AUTOSAR provides a General Specification on Basic Software modules [5, SWS BSW General], which is also valid for TTCAN Interface.

Thus, the specification SWS BSW General shall be considered as additional and required specification for `TtcanIf`.

4 Constraints and assumptions

The constraints and assumptions of TtcanIf are the same as for CanIf [2].

5 Dependencies to other modules

5.1 Additional TTCAN specific dependencies to other modules

This section describes the relations to other modules within the AUTOSAR basic software architecture. It contains brief descriptions of configuration information and services, which are additional required by TtcanIf from other modules. The dependencies described in the referenced CanIf [2] also apply for TtcanIf.

5.1.1 AUTOSAR Operating System

It's possible to use dedicated Job List Execution Functions (JLEF) for each TTCAN Controller.

Whether the optional JLEF runs in a task concept or in an ISR is implementation specific. Refer to section 7.3.

5.1.2 AUTOSAR PDU router

Additional to the data access through CanIf, as described in [2], TtcanIf can call a JLEF synchronously to the TTCAN Local Time. This shall ensure the request for data to be sent occur synchronously to the TTCAN Local Time. Within the JLEF TtcanIf calls the callback function <UL_TriggerTransmit> of PduR in order to start the copy operation of PDU data. Additionally the JLEF can be used to read out received data synchronoulsy to the TTCAN Local Time.

5.1.3 Upper Protocol Layers

Inside the AUTOSAR BSW architecture the Upper Layers (UL) of TtcanIf are represented by the PduR, CanNm, CanTp, CanSM, and EcuM.

If the respective upper layer BSW module does not operate synchronously to the TTCAN Local Time, all occurrences are asynchronous to the code execution of this BSW module.

5.1.4 TTCAN Driver

TtcanIf provides additional notification services used by TtcanDrv (refer to section 8.5).

6 Requirements Tracing

Requirement	Description	Satisfied by
[SRS_BSW_00337]	Classification of development errors	[SWS_TtCanIf_00007] [SWS_TtCanIf_00008] [SWS_TtCanIf_00145]
[SRS_Can_01121]	CAN Interface shall be the interface layer between the underlying CAN Driver(s) and CAN transceiver Driver(s) and Upper Layers	[SWS_TtCanIf_00065] [SWS_TtCanIf_00067] [SWS_TtCanIf_00069] [SWS_TtCanIf_00070] [SWS_TtCanIf_00072] [SWS_TtCanIf_00073] [SWS_TtCanIf_00074] [SWS_TtCanIf_00075] [SWS_TtCanIf_00076] [SWS_TtCanIf_00077] [SWS_TtCanIf_00080] [SWS_TtCanIf_00082] [SWS_TtCanIf_00083] [SWS_TtCanIf_00084] [SWS_TtCanIf_00085] [SWS_TtCanIf_00086] [SWS_TtCanIf_00087] [SWS_TtCanIf_00101] [SWS_TtCanIf_00102] [SWS_TtCanIf_00103] [SWS_TtCanIf_00104] [SWS_TtCanIf_00105] [SWS_TtCanIf_00106] [SWS_TtCanIf_00107] [SWS_TtCanIf_00108] [SWS_TtCanIf_00109] [SWS_TtCanIf_00110] [SWS_TtCanIf_00112] [SWS_TtCanIf_00113] [SWS_TtCanIf_00114] [SWS_TtCanIf_00115] [SWS_TtCanIf_00116] [SWS_TtCanIf_00117] [SWS_TtCanIf_00119]
[SRS_Can_01131]	The CAN Interface module shall provide the possibility to have polling and callback notification mechanism in parallel	[SWS_TtCanIf_00089] [SWS_TtCanIf_00090] [SWS_TtCanIf_00091] [SWS_TtCanIf_00092] [SWS_TtCanIf_00093] [SWS_TtCanIf_00094]
[SRS_TtCan_41010]	A Job List shall be configurable.	[SWS_TtCanIf_00002] [SWS_TtCanIf_00141] [SWS_TtCanIf_00143]

[SRS_TtCan_41011]	If a Job List is available (see SRS_Tt Can_41010) it shall be executed by a separate Job List Execution Function.	[SWS_TtCanIf_00004] [SWS_TtCanIf_00006] [SWS_TtCanIf_00007] [SWS_TtCanIf_00032] [SWS_TtCanIf_00033] [SWS_TtCanIf_00079] [SWS_TtCanIf_00145]
[SRS_TtCan_41013]	An occurred severe error (S3) shall be processed as a BusOff (see SRS_Can_01029 of CAN SRS)	[SWS_TtCanIf_00120] [SWS_TtCanIf_00121] [SWS_TtCanIf_00122]

7 Functional specification

7.1 General Functionality

Time-triggered CAN is a higher level protocol layer additional to the CAN protocol itself, which remains unchanged within the time-triggered communication.

This functional specification only provide specifications, which are additional to the CAN stack, to realize the mode Time Triggered CAN (TTCAN). Nevertheless the implementation shall provide the Standard CAN mode anyway.

7.2 TTCAN Interface State Machine

`TtcanIf` use the same states as `CanIf`.

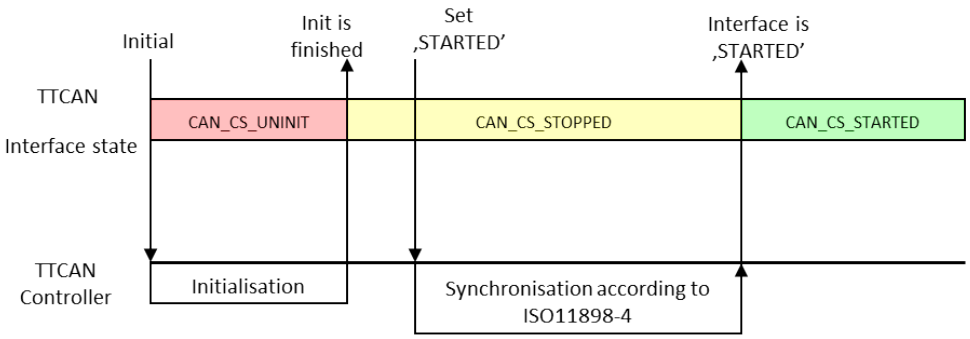

Figure 7.1: Exemplary Startup of TTCAN

7.3 TTCAN Job List

A `TTCAN Job List` is a list of `Communication Job`s sorted according to their respective execution start time.

The `TTCAN Job List` shall be used if a synchronized copy operation into the `Controller` is required and/or a synchronized readout of the `Controller` (optional feature) shall be realized. Otherwise the normal CAN procedure without a `Job List` can be used.

[SWS_TtCanIf_00002] ⌈ The Copy Operation into/from the `TTCAN Controller` shall be scheduled within a `Job List`. ⌋ *(SRS_TtCan_41010)*

[SWS_TtCanIf_00143] ⌈ For each `Controller` that is controlled by `TtcanIf` one dedicated `Job List` and one dedicated `JLEF` (refer to section 7.3) shall be used. It's possible to mixture both variants, with and without the usage of a `Job List`. ⌋ *(SRS_TtCan_41010)*

7.4 TTCAN Job List Execution Function

[SWS_TtCanIf_00004] ⌈ If a Job List is used, the TTCAN Job List Execution Function (JLEF) shall execute the Communication Jobs of the Job List synchronously to the Controller time (i.e. at well-defined points in time). ⌋ *(SRS_TtCan_41011)*

The execution of JLEF is implementation specific.

[SWS_TtCanIf_00006] ⌈ The API names of the JLEF shall obey the following pattern:

- `CanIf_TTJobListExec_0()` for Controller # 0
- `CanIf_TTJobListExec_1()` for Controller # 1
- `CanIf_TTJobListExec_2()` for Controller # 2
- `CanIf_TTJobListExec_3()` for Controller # 3
- ... and so on, if more than 4 Controllers are supported.

⌋*(SRS_TtCan_41011)*

[SWS_TtCanIf_00007] ⌈ If the JLEF lost synchronisation to the Local Time of the TTCAN Controller then the function `Dem_SetEventStatus(CANIF_TT_E_JLE_SYNC, DEM_EVENT_STATUS_FAILED)` shall be called. ⌋*(SRS_TtCan_41011, SRS_BSW_00337)*

[SWS_TtCanIf_00145] ⌈ If the JLEF was executed successfully, then the function `Dem_SetEventStatus(CANIF_TT_E_JLE_SYNC, DEM_EVENT_STATUS_PASSED)` shall be called. ⌋*(SRS_TtCan_41011, SRS_BSW_00337)*

Exemplary the JLEF performs the following steps:

1. Retrieve the cycle time of the Controller by calling `Can_TTGetControllerTime()`.
   - If the cycle time cannot be retrieved
     (a) Call `Dem_SetEventStatus(CANIF_TT_E_JLE_SYNC, DEM_EVENT_STATUS_FAILED)`
     (b) Terminate the execution of JLEF.
   - Otherwise, the JLEF continues with step 2.
2. Check whether the JLEF was called by start of new Basic cycle.
   - If it is false, continue with step 3.
   - Otherwise check whether the next job is scheduled for this Basic cycle.

- If it is TRUE, set the interrupt timer to the next job's start time in order to invoke the JLEF again and terminate the execution of JLEF
- Otherwise terminate execution of JLEF.

3. If the cycle Time delay compared to the job start time is larger than a maximum delay (configuration parameter CanIfTTMaxIsrDelay), the execution of the Job List is considered to be asynchronous to the local time and thus the following actions are performed:
   (a) Call Dem_SetEventStatus(CANIF_TT_E_JLE_SYNC, DEM_EVENT_STATUS_FAILED)
   (b) Add some 'safety margin' (i.e. some timespan which takes jitter into account)
   (c) Search the Job List for the subsequent job, i.e. that job with an invocation time greater than the current Local Time + safety margin.
   (d) Search for the next Job List entry, which is valid for the current Basic Cycle. If the end of the Job List is reached, wrap around to the next Basic Cycle and continue the search for that respective Basic Cycle.
   (e) If the next job is scheduled for this Basic Cycle:
   - Schedule next job, exemplary by using the time mark interrupt
   - Otherwise disable timer interrupt

   (f) Terminate the execution of JLEF.

   Otherwise, the JLEF continues with step 4.
4. Retrieve the sorted list of Communication Operations of the current Job pointed to by the current job pointer and execute the retrieved communication operations in the configured order.
5. Search for the next Job List entry, which is valid for the current Basic Cycle. If the end of the Job List is reached, wrap around to the next Basic Cycle and continue the search for that respective Basic Cycle.
6. If the next job is scheduled for this Basic cycle set the interrupt timer to this job's start time Otherwise disable timer interrupt
7. Call Dem_SetEventStatus(CANIF_TT_E_JLE_SYNC, DEM_EVENT_STATUS_PASSED)
8. Terminate the execution of JLEF.

7.5 Data communication via TTCAN

TTCAN is a deterministic time driven communication system. Each datum that should be transmitted or received has to be scheduled at system configuration time.

A detailed description of Synchronization, Transmission Triggering, Reception Triggering, Initialization and Failure handling can be found in [1, ISO 11898-4].

Additional TTCAN specific requirements:

[SWS_TtCanIf_00141] ⌈ If a Job List is configured for a Tx L-PDU (see CanIfTTJoblist), a function call of CanIf_Transmit() (see *SWS_CanIf_00318*) shall not directly call Can_Write(). The information that a call of CanIf_Transmit() occurred has to be buffered within TtcanIf until the data is transmitted by the Job List. ⌋*(SRS_TtCan_41010)*

Note: The kind of buffering the information of [SWS_TtCanIf_00141] is implementation specific.

Rationale for [SWS_TtCanIf_00141]: A Job List needs to be configured for HW Objects which transmit in *BasicCAN* mode, where one HW Object can be used to serve different time slots within the TTCAN system matrix. In this case a Job List has to take care, which message is available in the HW Object at the correct time. A Can_Write() call directly after CanIf_Transmit() can violate this.

7.6 TTCAN Controller mode

This chapter corresponds to the chapter "'CAN Controller mode"' of the [2, CAN Interface SWS].

[SWS_TtCanIf_00120] ⌈ If a CanIf Controller mode state machine is either in state CAN_CS_STARTED, CAN_CS_STOPPED or CAN_CS_SLEEP when function CanIf_TTSevereError() is called, then CanIf shall call the function CanSM_ControllerBusOff() for the CAN Network assigned to parameter Controller of CanIf_TTSevereError(). ⌋*(SRS_TtCan_41013)*

[SWS_TtCanIf_00121] ⌈ If a CanIf Controller mode state machine is in state CAN_CS_STARTED when the function CanIf_TTSevereError(ControllerId, CanIf_TTSevereError) is called with parameter ControllerId referencing that CanIf Controller mode state machine, then CanIf shall call Can_SetControllerMode(Controller, CAN_CS_STOPPED) and CanIf shall call CanSM_ControllerBusOff(ControllerId) of CanSM. ⌋*(SRS_TtCan_41013)*

These APIs are mapped to a BusOff API of CanSM, because, they indicate a severe error of the TTCAN Controller. The handling and recovery of such an error is equal to BusOff.

7.7 Error classification

7.7.1 Development Errors

There are no development errors.

7.7.2 Runtime Errors

There are no runtime errors.

7.7.3 Transient Faults

There are no transient faults.

7.7.4 Production Errors

There are no production errors.

7.7.5 Extended Production Errors

[SWS_TtCanIf_00008] ⌈ Extended Production Errors of `TtcanIf` are defined in 7.1. ⌋*(SRS_BSW_00337)*

Error Name:	`CANIF_TT_E_JLE_SYNC`	
Short Description:	Lost Synchronization	
Long Description:	`Job List Execution Function` lost synchronization to the `TTCAN Local Time`.	
Detection Criteria:	Fail	If the `JLEF` lost synchronization to the `Local Time` of the `TTCAN Controller` (see [SWS_TtCanIf_00007]), e.g.: • If the cycle time cannot be retrieved • If the cycle time delay compared to the job start time is larger than a maximum delay
	Pass	`JLEF` was executed without synchronization loss
Secondary Parameters:	-	
Time Required:	depends on cause (e.g. `CanIfTTMaxIsrDelay`)	
Monitor Frequency:	continuous (see [SWS_TtCanIf_00007])	

Table 7.1: Definition of Extended Production Errors

8 API specification

In the following sections, the TTCAN specific APIs and types are described.

8.1 Imported types

Additional TTCAN specific imported types

[SWS_TtCanIf_00124] ⌈

Module	*Imported Type*
Can	Can_TTErrorLevelType Can_TTMasterStateType Can_TTTURType Can_TTTimeSourceType Can_TTTimeType
Can_GeneralTypes	Can_IdType
ComStack_Types	PduIdType PduInfoType
Dem	Dem_EventIdType Dem_EventStatusType
Std_Types	Std_ReturnType

Table 8.1: TtcanIf_ImportedTypes

⌋()

Note: `PduIdType` is missing as of `ComStack_Types`.

8.2 Type definitions

Additional TTCAN specific type definitions

8.2.1 CanIf_TTTimeType

[SWS_TtCanIf_00059] ⌈

Name:	`CanIf_TTTimeType`
Type:	`uint16`
Description:	16 bit value representing time values of TTCAN, e.g. cycle, local or global time

Table 8.2: CanIf_TTTimeType

⌋()

8.2.2 CanIf_TTMasterSlaveModeType

[SWS_TtCanIf_00096] ⌈

Name:	CanIf_TTMasterSlaveModeType		
Type:	Enumeration		
Range:	CANIF_TT_BACKUP_MASTER	–	Master-Slave Mode: Backup master
	CANIF_TT_CURRENT_MASTER	–	Master-Slave Mode: Current master
	CANIF_TT_MASTER_OFF	–	Master-Slave Mode: Master off
	CANIF_TT_SLAVE	–	Master-Slave Mode: Slave
Description:	Master-Slave Mode		

Table 8.3: CanIf_TTMasterSlaveModeType

⌋()

8.2.3 CanIf_TTSyncModeEnumType

[SWS_TtCanIf_00097] ⌈

Name:	CanIf_TTSyncModeEnumType		
Type:	Enumeration		
Range:	CANIF_TT_IN_GAP	–	Sync mode: In_Gap
	CANIF_TT_IN_SCHEDULE	–	Sync mode: In_Schedule
	CANIF_TT_SYNC_OFF	–	Sync mode: Sync_Off
	CANIF_TT_SYNCHRONIZING	–	Sync mode: Synchronizing
Description:	Sync mode		

Table 8.4: CanIf_TTSyncModeEnumType

⌋()

8.2.4 CanIf_TTMasterStateType

[SWS_TtCanIf_00060] ⌈

Name:	CanIf_TTMasterStateType		
Type:	Structure		
Element:	CanIf_TTMaster SlaveModeType	masterSlaveMode	–
	uint8	refTriggerOffset	current value of ref trigger offset
	CanIf_TTSyncMode EnumType	syncMode	–
Description:	Master state type including sync mode, master-slave mode and current ref trigger offset		

Table 8.5: CanIf_TTMasterStateType

⌋()

8.2.5 CanIf_TTErrorLevelEnumType

[SWS_TtCanIf_00098] ⌈

Name:	CanIf_TTErrorLevelEnumType		
Type:	Enumeration		
Range:	CANIF_TT_ERROR_S0	–	Error level S0: No Error
	CANIF_TT_ERROR_S1	–	Error level S1: Warning
	CANIF_TT_ERROR_S2	–	Error level S2: Error
	CANIF_TT_ERROR_S3	–	Error level S3: Fatal Error
Description:	Error level (S0-S3)		

Table 8.6: CanIf_TTErrorLevelEnumType

⌋()

8.2.6 CanIf_TTErrorLevelType

[SWS_TtCanIf_00061] ⌈

Name:	CanIf_TTErrorLevelType		
Type:	Structure		
Element:	CanIf_TTErrorLevel EnumType	errorLevel	Error Level (S0-S3)
	uint8	maxMessageStatus Count	Max value of message status count (0-7)
	uint8	minMessageStatus Count	Min value of message status count (0-7)
Description:	TTCAN error level including min and max values of message status count		

Table 8.7: CanIf_TTErrorLevelType

⌋()

8.2.7 CanIf_TTSevereErrorEnumType

[SWS_TtCanIf_00137] ⌈

Name:	CanIf_TTSevereErrorEnumType		
Type:	Enumeration		
Range:	CANIF_TT_CONFIG_ERROR	–	Event: see ISO11898-4
	CANIF_TT_WATCH_TRIGGER_REACHED	–	Event: Watch Trigger reached
	CANIF_TT_APPL_WATCHDOG	–	Event: see ISO 11898-4
Description:	Event that causes a severe error		

Table 8.8: CanIf_TTSevereErrorEnumType

⌋()

8.2.8 CanIf_TTTimeSourceType

[SWS_TtCanIf_00063] ⌈

Name:	CanIf_TTTimeSourceType		
Type:	Enumeration		
Range:	CANIF_TT_CYCLE_TIME	–	Time source: Cycle Time
	CANIF_TT_GLOBAL_TIME	–	Time source: Global Time
	CANIF_TT_LOCAL_TIME	–	Time source: Local Time
	CANIF_TT_UNDEFINED	–	Time source: Undefined
Description:	Time source of time values in TTCAN		

Table 8.9: CanIf_TTTimeSourceType

⌋()

8.2.9 CanIf_TTEventEnumType

[SWS_TtCanIf_00099] ⌈

Name:	CanIf_TTEventEnumType		
Type:	Enumeration		
Range:	CANIF_TT_ERROR_LEVEL_CHANGED	–	Event: Error Level changed
	CANIF_TT_INIT_WATCH_TRIGGER	–	Event: Init Watch Trigger reached
	CANIF_TT_NO_ERROR	–	No error
	CANIF_TT_SYNC_FAILED	–	Event: Sync failed
	CANIF_TT_TX_OVERFLOW	–	Event: Tx Overflow
	CANIF_TT_TX_UNDERFLOW	–	Event: Tx Underflow
Description:	Event that causes a Timing/Error IRQ		

Table 8.10: CanIf_TTEventEnumType

⌋()

8.2.10 CanIf_TTTimingErrorIRQType

[SWS_TtCanIf_00064] ⌈

Name:	CanIf_TTTimingErrorIRQType		
Type:	Structure		
Element:	CanIf_TTErrorLevelType	errorLevel	Current error level

	CanIf_TTEventEnumType	event	Event that caused the IRQ
Description:	Combines all events that are reported by CanIf_TTTimingError (event indication and error level)		

Table 8.11: CanIf_TTTimingErrorIRQType

⌋()

8.3 Function definitions

Additional TTCAN specific function definitions

8.3.1 CanIf_TTGetControllerTime

[SWS_TtCanIf_00065] ⌈

Service name:	CanIf_TTGetControllerTime	
Syntax:	`Std_ReturnType CanIf_TTGetControllerTime( uint8 ControllerId, CanIf_TTTimeType* CanIf_TTGlobalTime, CanIf_TTTimeType* CanIf_TTLocalTime, CanIf_TTTimeType* CanIf_TTCycleTime, uint8* CanIf_TTCycleCount )`	
Service ID[hex]:	0x33	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	ControllerId	Controller from which the time information shall be retrieved
Parameters (inout):	None	
Parameters (out):	CanIf_TTGlobalTime CanIf_TTLocalTime CanIf_TTCycleTime CanIf_TTCycleCount	Address to store return value: Global time Address to store return value: Local time Address to store return value: Cycle time Address to store return value: Cycle count value
Return value:	Std_ReturnType	E_OK: Function successful E_NOT_OK: Development error occurred
Description:	Gets the current values for the global, local and cycle time and the cycle count of the controller	

Table 8.12: CanIf_TTGetControllerTime

⌋(*SRS_Can_01121*)

[SWS_TtCanIf_00101] ⌈ The function `CanIf_TTGetControllerTime()` shall call `Can_TTGetControllerTime(Controller, Can_TTGlobalTime, CanTTLocalTime, Can_TTCycleTime, Can_TTCycleCount).` ⌋(*SRS_Can_01121*)

[SWS_TtCanIf_00010] ⌈ If parameter `Controller` of `CanIf_TTGetControllerTime()` has an invalid value and if development error detection is enabled (i.e. `CANIF_DEV_ERROR_DETECT` equals `ON`), the function `CanIf_TTGetControllerTime()` shall report development error code `CANIF_E_PARAM_CONTROLLER` to the `Det_ReportError` service of the DET module. ⌋*()*

[SWS_TtCanIf_00011] ⌈ Caveats of `CanIf_TTGetControllerTime()`: `TtcanIf` has to be initialized before this API service may be called. ⌋*()*

[SWS_TtCanIf_00066] ⌈ If development error detection for `TtcanIf` is enabled: The function `CanIf_TTGetControllerTime()` shall raise the error `CANIF_E_PARAM_POINTER` and shall return `E_NOT_OK` if one of the parameter `CanIf_TTCycleCount`, `CanIf_TTGlobalTime`, `CanIf_TTLocalTime` and `CanIf_TTCycleTime` is a `NULL` pointer. ⌋*()*

8.3.2 CanIf_TTGetMasterState

[SWS_TtCanIf_00067] ⌈

Service name:	CanIf_TTGetMasterState	
Syntax:	`Std_ReturnType CanIf_TTGetMasterState( uint8 ControllerId, CanIf_TTMasterStateType* CanIf_TTMasterState )`	
Service ID[hex]:	0x34	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN controller
Parameters (inout):	None	
Parameters (out):	CanIf_TTMaster State	Address to store return value: Master state
Return value:	Std_ReturnType	E_OK: Function successful E_NOT_OK: Development error occurred
Description:	Gets the master state. The master state includes the sync mode (sync_off, synchronizing, in_gap, in_schedule) the master-slave mode (master_off, slave, backup_master, current_master) and the current value for ref trigger offset.	

Table 8.13: CanIf_TTGetMasterState

⌋*(SRS_Can_01121)*

[SWS_TtCanIf_00102] ⌈ The function `CanIf_TTGetMasterState()` shall call `Can_TTGetMasterState(Controller, Can_TTMasterState).` ⌋ *(SRS_Can_01121)*

[SWS_TtCanIf_00012] ⌈ If parameter `Controller` of `CanIf_TTGetMasterState()` has an invalid value and if development er-

ror detection is enabled (i.e. CANIF_DEV_ERROR_DETECT equals ON), the function CanIf_TTGetMasterState() shall report development error code CANIF_E_PARAM_CONTROLLER to the Det_ReportError service of the DET module. ⌋()

[SWS_TtCanIf_00013] ⌈ Caveats of CanIf_TTGetMasterState(): TtcanIf has to be initialized before this API service may be called. ⌋()

[SWS_TtCanIf_00068] ⌈ If development error detection for TtcanIf is enabled: The function CanIf_TTGetMasterState() shall raise the error CAN_E_PARAM_POINTER and shall return E_NOT_OK if the parameter CanIf_TTMasterState is a NULL pointer. ⌋()

8.3.3 CanIf_TTGetNTUActual

[SWS_TtCanIf_00069] ⌈

Service name:	CanIf_TTGetNTUActual	
Syntax:	Std_ReturnType CanIf_TTGetNTUActual(uint8 ControllerId, float32 CanIf_TTNTUAct)	
Service ID[hex]:	0x35	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN controller
Parameters (inout):	None	
Parameters (out):	CanIf_TTNTUAct	Address to store return value: Actual value of NTU. Value is given in microseconds
Return value:	Std_ReturnType	E_OK: Function successful E_NOT_OK: Development error occurred
Description:	Gets the actual value of NTU (network time unit). Together with the local oscillator period, the actual value of NTU can be derived from the actual value of TUR.	

Table 8.14: CanIf_TTGetNTUActual

⌋(*SRS_Can_01121*)

[SWS_TtCanIf_00103] ⌈ The function CanIf_TTGetNTUActual() shall call Can_TTGetNTUActual(Controller, Can_TTTURAct). ⌋(*SRS_Can_01121*)

[SWS_TtCanIf_00014] ⌈ If parameter Controller of CanIf_TTGetNTUActual() has an invalid value and if development error detection is enabled (i.e. CANIF_DEV_ERROR_DETECT equals ON), the function CanIf_TTGetNTUActual() shall report development error code CANIF_E_PARAM_CONTROLLER to the Det_ReportError service of the DET module. ⌋()

[SWS_TtCanIf_00015] ⌈ Caveats of `CanIf_TTGetNTUActual()`: `TtcanIf` has to be initialized before this API service may be called. ⌋*()*

8.3.4 CanIf_TTGetErrorLevel

[SWS_TtCanIf_00070] ⌈

Service name:	CanIf_TTGetErrorLevel	
Syntax:	`Std_ReturnType CanIf_TTGetErrorLevel( uint8 ControllerId, CanIf_TTErrorLevelType* CanIf_TTErrorLevel )`	
Service ID[hex]:	0x36	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN controller from which the error level shall be retrieved
Parameters (inout):	None	
Parameters (out):	CanIf_TTErrorLevel	Address to store return value: Error level
Return value:	Std_ReturnType	E_OK: Function successful E_NOT_OK: Development error occurred
Description:	Gets the error level. This includes the severity of the error level (S0-S3) and the minimum and maximum value of the message status count.	

Table 8.15: CanIf_TTGetErrorLevel

⌋*(SRS_Can_01121)*

[SWS_TtCanIf_00104] ⌈ The function `CanIf_TTGetErrorLevel()` shall call `Can_TTGetErrorLevel(Controller, Can_TTErrorLevel).` ⌋ *(SRS_Can_01121)*

[SWS_TtCanIf_00016] ⌈ If parameter `Controller` of `CanIf_TTGetErrorLevel()` has an invalid value and if development error detection is enabled (i.e. `CANIF_DEV_ERROR_DETECT` equals `ON`), the function `CanIf_TTGetErrorLevel()` shall report development error code `CANIF_E_PARAM_CONTROLLER` to the `Det_ReportError` service of the DET module. ⌋*()*

[SWS_TtCanIf_00017] ⌈ Caveats of `CanIf_TTGetErrorLevel()`: `TtcanIf` has to be initialized before this API service may be called. ⌋*()*

[SWS_TtCanIf_00071] ⌈ If development error detection for `TtcanIf` is enabled: The function `CanIf_TTGetErrorLevel()` shall raise the error `CAN_E_PARAM_POINTER` and shall return `E_NOT_OK` if the parameter `CanIf_TTErrorLevel` is a `NULL` pointer. ⌋*()*

8.3.5 CanIf_TTSetNextIsGap

[SWS_TtCanIf_00072] ⌈

Service name:	CanIf_TTSetNextIsGap	
Syntax:	`Std_ReturnType CanIf_TTSetNextIsGap( uint8 ControllerId )`	
Service ID[hex]:	0x37	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN controller
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Function successful E_NOT_OK: Development error occurred
Description:	Sets the "Next_is_Gap" bit.	

Table 8.16: CanIf_TTSetNextIsGap

⌋(*SRS_Can_01121*)

[SWS_TtCanIf_00105] ⌈ The function `CanIf_TTSetNextIsGap()` shall call `Can_TTSetNextIsGap(Controller)`.⌋(*SRS_Can_01121*)

[SWS_TtCanIf_00018] ⌈ If parameter `Controller` of `CanIf_TTSetNextIsGap()` has an invalid value and if development error detection is enabled (i.e. `CANIF_DEV_ERROR_DETECT` equals `ON`), the function `CanIf_TTSetNextIsGap()` shall report development error code `CANIF_E_PARAM_CONTROLLER` to the `Det_ReportError` service of the DET module. ⌋()

[SWS_TtCanIf_00019] ⌈ Caveats of `CanIf_TTSetNextIsGap()`: `TtcanIf` has to be initialized before this API service may be called. ⌋()

8.3.6 CanIf_TTSetEndOfGap

[SWS_TtCanIf_00073] ⌈

Service name:	CanIf_TTSetEndOfGap	
Syntax:	`Std_ReturnType CanIf_TTSetEndOfGap( uint8 ControllerId )`	
Service ID[hex]:	0x38	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN controller
Parameters (inout):	None	

Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Function successful E_NOT_OK: Development error occurred
Description:	Signals the end of a gap.	

Table 8.17: CanIf_TTSetEndOfGap

⌋(*SRS_Can_01121*)

[SWS_TtCanIf_00106] ⌈ The function `CanIf_TTSetEndOfGap()` shall call `Can_TTSetNextIsGap(Controller)`. ⌋(*SRS_Can_01121*)

[SWS_TtCanIf_00020] ⌈ If parameter `Controller` of `CanIf_TTSetEndOfGap()` has an invalid value and if development error detection is enabled (i.e. `CANIF_DEV_ERROR_DETECT` equals `ON`), the function `CanIf_TTSetEndOfGap()` shall report development error code `CANIF_E_PARAM_CONTROLLER` to the `Det_ReportError` service of the DET module. ⌋()

[SWS_TtCanIf_00021] ⌈ Caveats of `CanIf_TTSetEndOfGap()`: `TtcanIf` has to be initialized before this API service may be called. ⌋()

8.3.7 CanIf_TTSetTimeCommand

[SWS_TtCanIf_00074] ⌈

Service name:	CanIf_TTSetTimeCommand	

Syntax:	`Std_ReturnType CanIf_TTSetTimeCommand(`
	`uint8 ControllerId`
	`)`	

Service ID[hex]:	0x39	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN controller
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Function successful E_NOT_OK: Development error occurred
Description:	Adjusts the global time at the beginning of the next basic cycle by the amount of "global time preset"	

Table 8.18: CanIf_TTSetTimeCommand

⌋(*SRS_Can_01121*)

[SWS_TtCanIf_00107] ⌈ The function `CanIf_TTSetTimeCommand()` shall call `Can_TTSetTimeCommand(Controller)`. ⌋(*SRS_Can_01121*)

[SWS_TtCanIf_00022] ⌈ If parameter `Controller` of `CanIf_TTSetTimeCommand()` has an invalid value and if development error detection is enabled (i.e. `CANIF_DEV_ERROR_DETECT` equals `ON`), the function `CanIf_TTSetTimeCommand()` shall report development error code `CANIF_E_PARAM_CONTROLLER` to the `Det_ReportError` service of the DET module. ⌋*()*

[SWS_TtCanIf_00023] ⌈ Caveats of `CanIf_TTSetTimeCommand()`: `TtcanIf` has to be initialized before this API service may be called. ⌋*()*

8.3.8 CanIf_TTGlobalTimePreset

[SWS_TtCanIf_00075] ⌈

Service name:	CanIf_TTGlobalTimePreset	
Syntax:	`Std_ReturnType CanIf_TTGlobalTimePreset( uint8 ControllerId, CanIf_TTTimeType CanIf_TTGlobalTimePreset )`	
Service ID[hex]:	0x3a	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	ControllerId CanIf_TTGlobal TimePreset	Abstracted CanIf ControllerId which is assigned to a CAN controller New value for "global time preset"
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Function successful E_NOT_OK: Development error occurred
Description:	Sets the value of "global time preset".	

Table 8.19: CanIf_TTGlobalTimePreset

⌋*(SRS_Can_01121)*

[SWS_TtCanIf_00108] ⌈ The function `CanIf_TTGlobalTimePreset()` shall call `Can_TTGlobalTimePreset(Controller, Can_TTGlobalTimePreset)`. ⌋ *(SRS_Can_01121)*

[SWS_TtCanIf_00024] ⌈ If parameter `Controller` of `CanIf_TTGlobalTimePreset()` has an invalid value and if development error detection is enabled (i.e. `CANIF_DEV_ERROR_DETECT` equals `ON`), the function `CanIf_TTGlobalTimePreset()` shall report development error code `CANIF_E_PARAM_CONTROLLER` to the `Det_ReportError` service of the DET module. ⌋*()*

[SWS_TtCanIf_00025] ⌈ Caveats of `CanIf_TTGlobalTimePreset()`: `TtcanIf` has to be initialized before this API service may be called. ⌋*()*

8.3.9 CanIf_TTSetExtClockSyncCommand

[SWS_TtCanIf_00076] ⌈

Service name:	CanIf_TTSetExtClockSyncCommand	
Syntax:	`Std_ReturnType CanIf_TTSetExtClockSyncCommand( uint8 ControllerId )`	
Service ID[hex]:	0x3b	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN controller
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Function successful E_NOT_OK: Development error occurred
Description:	Adjusts the NTU (network time unit) according to the value given by "NTU adjust". Together with the local oscillator period, "TUR adjust" can be derived from "NTU adjust".	

Table 8.20: CanIf_TTSetExtClockSyncCommand

⌋*(SRS_Can_01121)*

[SWS_TtCanIf_00109] ⌈ The function `CanIf_TTSetExtClockSyncCommand()` shall call `Can_TTSetExtClockSyncCommand(Controller)`. ⌋*(SRS_Can_01121)*

[SWS_TtCanIf_00026] ⌈ If parameter `Controller` of `CanIf_TTSetExtClockSyncCommand()` has an invalid value and if development error detection is enabled (i.e. `CANIF_DEV_ERROR_DETECT` equals `ON`), the function `CanIf_TTSetExtClockSyncCommand()` shall report development error code `CANIF_E_PARAM_CONTROLLER` to the `Det_ReportError` service of the DET module. ⌋*()*

[SWS_TtCanIf_00027] ⌈ Caveats of `CanIf_TTSetExtClockSyncCommand()`: `TtcanIf` has to be initialized before this API service may be called. ⌋*()*

8.3.10 CanIf_TTSetNTUAdjust

[SWS_TtCanIf_00077] ⌈

Service name:	CanIf_TTSetNTUAdjust
Syntax:	`Std_ReturnType CanIf_TTSetNTUAdjust( uint8 ControllerId, float32 CanIf_TTNTUAdjust )`
Service ID[hex]:	0x3c
Sync/Async:	Synchronous

Reentrancy:	Non Reentrant	
Parameters (in):	ControllerId CanIf_TTNTUAdjust	Abstracted CanIf ControllerId which is assigned to a CAN controller New value for "NTU adjust". Value is given in microseconds.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Function successful E_NOT_OK: Development error occurred
Description:	Sets the value of "NTU adjust". Together with the local oscillator period, "TUR adjust" can be derived from "NTU adjust".	

Table 8.21: CanIf_TTSetNTUAdjust

⌋(*SRS_Can_01121*)

[SWS_TtCanIf_00110] ⌈ The function `CanIf_TTSetNTUAdjust()` shall call `Can_TTSetNTUAdjust(Controller, Can_TTNTUAdjust)`. ⌋(*SRS_Can_01121*)

[SWS_TtCanIf_00028] ⌈ If parameter `Controller` of `CanIf_TTSetNTUAdjust()` has an invalid value and if development error detection is enabled (i.e. `CANIF_DEV_ERROR_DETECT` equals `ON`), the function `CanIf_TTSetNTUAdjust()` shall report development error code `CANIF_E_PARAM_CONTROLLER` to the `Det_ReportError` service of the DET module. ⌋()

[SWS_TtCanIf_00029] ⌈ Caveats of `CanIf_TTSetNTUAdjust()`: `TtcanIf` has to be initialized before this API service may be called. ⌋()

8.4 Optional Function definitions

Additional optional TTCAN specific function definitions

8.4.1 CanIf_TTJobListExec_<Controller>

[SWS_TtCanIf_00079] ⌈

Service name:	CanIf_TTJobListExec_<Controller>

Syntax:	`void CanIf_TTJobListExec_<Controller>(`
	`void`
	`)`

Service ID[hex]:	0x50
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None

Return value:	None
Description:	Processes the job list of the TTCAN controller <Controller>.

Table 8.22: CanIf_TTJobListExec_<Controller>

⌋*(SRS_TtCan_41011)*

[SWS_TtCanIf_00032] ⌈ The function `CanIf_TTJobListExec_<Controller>()` shall exist once per `TTCAN Controller`, which use a `Job List`. ⌋ *(SRS_TtCan_41011)*

[SWS_TtCanIf_00033] ⌈ The function name of each instance of `CanIf_TTJobListExec_<Controller>()` shall contain the index of the respective `TTCAN Controller`. ⌋*(SRS_TtCan_41011)*

[SWS_TtCanIf_00034] ⌈ Caveats of `CanIf_TTJobListExec_<Controller>()`: `TtcanIf` has to be initialized before this API service may be called. ⌋*()*

For each `TTCAN Controller` (identified by index `Controller`), the execution of `CanIf_TTJobListExec_<Controller>()` can either run in a regular OS task or it is registered in the `AUTOSAR OS` as `ISR`, triggered by the `TTCAN Controller`.

8.4.2 CanIf_TTGetSyncQuality

[SWS_TtCanIf_00080] ⌈

Service name:	CanIf_TTGetSyncQuality	
Syntax:	`Std_ReturnType CanIf_TTGetSyncQuality( uint8 ControllerId, boolean* CanIf_TTClockSpeed, boolean* CanIf_TTGlobalTimePhase )`	
Service ID[hex]:	0x47	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN controller
Parameters (inout):	None	
Parameters (out):	CanIf_TTClock Speed CanIf_TTGlobal TimePhase	Address to store return value: True if the synchronization deviation is smaller than the "Synchronization deviation limit" Address to store return value: True if the the global time is in phase with the time master.
Return value:	Std_ReturnType	E_OK: Function successful E_NOT_OK: Development error occurred
Description:	Gets the synchronization quality.	

Table 8.23: CanIf_TTGetSyncQuality

⌋(*SRS_Can_01121*)

[SWS_TtCanIf_00112] ⌈ The function `CanIf_TTGetSyncQuality()` shall call `Can_TTGetSyncQuality(Controller, Can_TTClockSpeed, Can_TTGlobalTimePhase)`. ⌋(*SRS_Can_01121*)

[SWS_TtCanIf_00035] ⌈ If parameter `Controller` of `CanIf_TTGetSyncQuality()` has an invalid value and if development error detection is enabled (i.e. `CANIF_DEV_ERROR_DETECT` equals `ON`), the function `CanIf_TTGetSyncQuality()` shall report development error code `CANIF_E_PARAM_CONTROLLER` to the `Det_ReportError` service of the DET module. ⌋()

[SWS_TtCanIf_00036] ⌈ Caveats of `CanIf_TTGetSyncQuality()`: `TtcanIf` has to be initialized before this API service may be called. ⌋()

[SWS_TtCanIf_00081] ⌈ If development error detection for `TtcanIf` is enabled: The function `CanIf_TTGetSyncQuality()` shall raise the error `CAN_E_PARAM_POINTER` and shall return `E_NOT_OK` if one of the parameter `CanIf_ClockSpeed` and `CanIf_GlobalTimePhase` is a `NULL` pointer. ⌋()

8.4.3 CanIf_TTSetTimeMark

[SWS_TtCanIf_00082] ⌈

Service name:	CanIf_TTSetTimeMark	
Syntax:	`Std_ReturnType CanIf_TTSetTimeMark( uint8 ControllerId, CanIf_TTTimeType CanIf_TTTimeMark, CanIf_TTTimeSourceType CanIf_TTTimeSource )`	
Service ID[hex]:	0x48	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	ControllerId CanIf_TTTimeMark CanIf_TTTime Source	Abstracted CanIf ControllerId which is assigned to a CAN controller Gives the value of the time mark to be set. Defines the time source for the time mark to be set.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Function successful E_NOT_OK: Development error occurred
Description:	Sets a new value for the time mark for the given time source.	

Table 8.24: CanIf_TTSetTimeMark

⌋(*SRS_Can_01121*)

[SWS_TtCanIf_00113] ⌈ The function `CanIf_TTSetTimeMark()` shall call `Can_TTSetTimeMark(Controller, Can_TTTimeMark, Can_TTTimeSource)`.⌋*(SRS_Can_01121)*

[SWS_TtCanIf_00037] ⌈ If parameter `Controller` of `CanIf_TTSetTimeMark()` has an invalid value and if development error detection is enabled (i.e. `CANIF_DEV_ERROR_DETECT` equals `ON`), the function `CanIf_TTSetTimeMark()` shall report development error code `CANIF_E_PARAM_CONTROLLER` to the `Det_ReportError` service of the DET module. ⌋*()*

[SWS_TtCanIf_00038] ⌈ Caveats of `CanIf_TTSetTimeMark()`: `TtcanIf` has to be initialized before this API service may be called. ⌋*()*

8.4.4 CanIf_TTCancelTimeMark

[SWS_TtCanIf_00083] ⌈

Service name:	CanIf_TTCancelTimeMark	
Syntax:	`Std_ReturnType CanIf_TTCancelTimeMark( uint8 ControllerId )`	
Service ID[hex]:	0x49	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN controller
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Function successful E_NOT_OK: Development error occurred
Description:	Cancels the time mark.	

Table 8.25: CanIf_TTCancelTimeMark

⌋*(SRS_Can_01121)*

[SWS_TtCanIf_00114] ⌈ The function `CanIf_TTCancelTimeMark()` shall call `Can_TTCancelTimeMark(Controller)`.⌋*(SRS_Can_01121)*

[SWS_TtCanIf_00039] ⌈ If parameter `Controller` of `CanIf_TTCancelTimeMark()` has an invalid value and if development error detection is enabled (i.e. `CANIF_DEV_ERROR_DETECT` equals `ON`), the function `CanIf_TTCancelTimeMark()` shall report development error code `CANIF_E_PARAM_CONTROLLER` to the `Det_ReportError` service of the DET module. ⌋*()*

[SWS_TtCanIf_00040] ⌈ Caveats of `CanIf_TTCancelTimeMark()`: `TtcanIf` has to be initialized before this API service may be called. ⌋*()*

8.4.5 CanIf_TTAckTimeMark

[SWS_TtCanIf_00084] ⌈

Service name:	CanIf_TTAckTimeMark	
Syntax:	`Std_ReturnType CanIf_TTAckTimeMark( uint8 ControllerId )`	
Service ID[hex]:	0x4a	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN controller
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Function successful E_NOT_OK: Development error occurred
Description:	Acknowledges the time mark interrupt by resetting the flag in the interrupt vector register.	

Table 8.26: CanIf_TTAckTimeMark

⌋*(SRS_Can_01121)*

[SWS_TtCanIf_00115] ⌈ The function `CanIf_TTAckTimeMark()` shall call `Can_TTAckTimeMark(Controller)`. ⌋*(SRS_Can_01121)*

[SWS_TtCanIf_00041] ⌈ If parameter `Controller` of `CanIf_TTAckTimeMark()` has an invalid value and if development error detection is enabled (i.e. `CANIF_DEV_ERROR_DETECT` equals `ON`), the function `CanIf_TTAckTimeMark()` shall report development error code `CANIF_E_PARAM_CONTROLLER` to the `Det_ReportError` service of the DET module. ⌋*()*

[SWS_TtCanIf_00042] ⌈ Caveats of `CanIf_TTAckTimeMark()`: `TtcanIf` has to be initialized before this API service may be called. ⌋*()*

8.4.6 CanIf_TTEnableTimeMarkIRQ

[SWS_TtCanIf_00085] ⌈

Service name:	CanIf_TTEnableTimeMarkIRQ	
Syntax:	`Std_ReturnType CanIf_TTEnableTimeMarkIRQ( uint8 ControllerId )`	
Service ID[hex]:	0x4b	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN controller

Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Function successful E_NOT_OK: Development error occurred
Description:	Enables the time mark interrupt.	

Table 8.27: CanIf_TTEnableTimeMarkIRQ

⌋(*SRS_Can_01121*)

[SWS_TtCanIf_00116] ⌈ The function `CanIf_TTEnableTimeMarkIRQ()` shall call `Can_TTEnableTimeMarkIRQ(Controller)`. ⌋(*SRS_Can_01121*)

[SWS_TtCanIf_00043] ⌈ If parameter `Controller` of `CanIf_TTEnableTimeMarkIRQ()` has an invalid value and if development error detection is enabled (i.e. `CANIF_DEV_ERROR_DETECT` equals `ON`), the function `CanIf_TTEnableTimeMarkIRQ()` shall report development error code `CANIF_E_PARAM_CONTROLLER` to the `Det_ReportError` service of the DET module. ⌋()

[SWS_TtCanIf_00044] ⌈ Caveats of `CanIf_TTEnableTimeMarkIRQ()`: `TtcanIf` has to be initialized before this API service may be called. ⌋()

8.4.7 CanIf_TTDisableTimeMarkIRQ

[SWS_TtCanIf_00086] ⌈

Service name:	CanIf_TTDisableTimeMarkIRQ	

Syntax:	`Std_ReturnType CanIf_TTDisableTimeMarkIRQ(`
	`uint8 ControllerId`
	`)`	

Service ID[hex]:	0x4c	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN controller
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Function successful E_NOT_OK: Development error occurred
Description:	Disables the time mark interrupt.	

Table 8.28: CanIf_TTDisableTimeMarkIRQ

⌋(*SRS_Can_01121*)

[SWS_TtCanIf_00117] ⌈ The function `CanIf_TTDisableTimeMarkIRQ()` shall call `Can_TTDisableTimeMarkIRQ(Controller)`. ⌋(*SRS_Can_01121*)

[SWS_TtCanIf_00045] ⌈ If parameter `Controller` of `CanIf_TTDisableTimeMarkIRQ()` has an invalid value and if development error detection is enabled (i.e. `CANIF_DEV_ERROR_DETECT` equals `ON`), the function `CanIf_TTDisableTimeMarkIRQ()` shall report development error code `CANIF_E_PARAM_CONTROLLER` to the `Det_ReportError` service of the DET module. ⌋*()*

[SWS_TtCanIf_00046] ⌈ Caveats of `CanIf_TTDisableTimeMarkIRQ()`: `TtcanIf` has to be initialized before this API service may be called. ⌋*()*

8.4.8 CanIf_TTGetTimeMarkIRQStatus

[SWS_TtCanIf_00087] ⌈

Service name:	CanIf_TTGetTimeMarkIRQStatus	
Syntax:	`Std_ReturnType CanIf_TTGetTimeMarkIRQStatus( uint8 ControllerId, boolean* CanIf_TTIRQStatus )`	
Service ID[hex]:	0x4d	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN controller
Parameters (inout):	None	
Parameters (out):	CanIf_TTIRQStatus	Address to store return value: True if the timer for the time mark is pending.
Return value:	Std_ReturnType	E_OK: Function successful E_NOT_OK: Development error occurred
Description:	Gets the IRQ status of the time mark.	

Table 8.29: CanIf_TTGetTimeMarkIRQStatus

⌋*(SRS_Can_01121)*

[SWS_TtCanIf_00119] ⌈ The function `CanIf_TTGetTimeMarkIRQStatus()` shall call `Can_TTGetTimeMarkIRQStatus(Controller, Can_TTIRQStatus).` ⌋ *(SRS_Can_01121)*

[SWS_TtCanIf_00047] ⌈ If parameter `Controller` of `CanIf_TTGetTimeMarkIRQStatus()` has an invalid value and if development error detection is enabled (i.e. `CANIF_DEV_ERROR_DETECT` equals `ON`), the function `CanIf_TTGetTimeMarkIRQStatus()` shall report development error code `CANIF_E_PARAM_CONTROLLER` to the `Det_ReportError` service of the DET module. ⌋*()*

[SWS_TtCanIf_00048] ⌈ Caveats of `CanIf_TTGetTimeMarkIRQStatus()`: `TtcanIf` has to be initialized before this API service may be called. ⌋*()*

[SWS_TtCanIf_00088] ⌈ If development error detection for TtcanIf is enabled: The function CanIf_TTGetTimeMarkIRQStatus() shall raise the error CAN_E_PARAM_POINTER and shall return E_NOT_OK if the parameter CanIf_IRQStatus is a NULL pointer. ⌋*()*

8.5 Scheduled Functions

Additional TTCAN specific function definitions

TtcanIf has no additional scheduled functions.

8.6 Callback Notifications

This is a list of functions provided for other modules.

Additional TTCAN specific callback notifications

The callback notification specified within this chapter will be called by the CAN Driver module either in context of a main function or an interrupt.

8.6.1 CanIf_TTApplWatchdogError

[SWS_TtCanIf_00089] ⌈

Service name:	CanIf_TTApplWatchdogError	
Syntax:	`Std_ReturnType CanIf_TTApplWatchdogError( uint8 ControllerId )`	
Service ID[hex]:	0x5b	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN controller for which the application watchdog error shall be reported.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Function successful E_NOT_OK: Development error occurred
Description:	Reports an application watchdog error.	

Table 8.30: CanIf_TTApplWatchdogError

⌋*(SRS_Can_01131)*

[SWS_TtCanIf_00050] ⌈ If parameter ControllerId of CanIf_TTApplWatchdogError() has an invalid value and if development er-

ror detection is enabled (i.e. CANIF_DEV_ERROR_DETECT equals ON), then the function CanIf_TTApplWatchdogError() shall report development error code CANIF_E_PARAM_CONTROLLER to the Det_ReportError service of the DET module. ⌋()

8.6.2 CanIf_TTTimingError

[SWS_TtCanIf_00090] ⌈

Service name:	CanIf_TTTimingError	
Syntax:	`Std_ReturnType CanIf_TTTimingError( uint8 ControllerId, CanIf_TTTimingErrorIRQType CanIf_TTTimingErrorIRQ )`	
Service ID[hex]:	0x5c	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN controller for which the timing error shall be reported.
	CanIf_TTTimingErrorIRQ	Type of timing error.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Function successful E_NOT_OK: Development error occurred
Description:	Reports one of the following errors: - Change of error level - Tx overflow / underflow - Synchronization failed - Init watch trigger	

Table 8.31: CanIf_TTTimingError

⌋(*SRS_Can_01131*)

Note: This callback service is called by the CAN Driver module (supporting TTCAN) and implemented in the CAN Interface module (supporting TTCAN). It is called if error level S1 or S2 (see [1, ISO 11898-4]) have been detected in the corresponding controller.

[SWS_TtCanIf_00051] ⌈ If parameter ControllerId of CanIf_TTTimingError() has an invalid value and if development error detection is enabled (i.e. CANIF_DEV_ERROR_DETECT equals ON), then the function CanIf_TTTimingError() shall report development error code CANIF_E_PARAM_CONTROLLER to the Det_ReportError service of the DET module. ⌋()

8.6.3 CanIf_TTSevereError

[SWS_TtCanIf_00122] ⌈

Service name:	CanIf_TTSevereError	
Syntax:	`void CanIf_TTSevereError( uint8 ControllerId, CanIf_TTSevereErrorEnumType CanIf_TTSevereError )`	
Service ID[hex]:	0x5c	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN controller at which the severe error occured
	CanIf_TTSevere Error	type of severe error
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Reports one of the following errors: - failed to serve appl. watchdog - config error - watch trigger reached	

Table 8.32: CanIf_TTSevereError

⌋*(SRS_TtCan_41013)*

Note: This callback service is called by the CAN Driver module (supporting TTCAN) and implemented in the CAN Interface module (supporting TTCAN). It is called if error level S3 (severe error, see [1, ISO 11898-4]) has been detected in the corresponding controller.

[SWS_TtCanIf_00123] ⌈ If parameter `ControllerId` of `CanIf_TTSevereError()` has an invalid value and if development error detection is enabled (i.e. `CANIF_DEV_ERROR_DETECT` equals `ON`), then the function `CanIf_TTSevereError()` shall report development error code `CANIF_E_PARAM_CONTROLLER` to the `Det_ReportError` service of the DET module. ⌋*()*

8.6.4 CanIf_TTGap

[SWS_TtCanIf_00091] ⌈

Service name:	CanIf_TTGap
Syntax:	`Std_ReturnType CanIf_TTGap( uint8 ControllerId )`
Service ID[hex]:	0x5d
Sync/Async:	Synchronous

Reentrancy:	Non Reentrant	
Parameters (in):	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN controller for which the gap shall be reported.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Function successful E_NOT_OK: Development error occurred
Description:	Reports the occurrence of a gap.	

Table 8.33: CanIf_TTGap

⌋(*SRS_Can_01131*)

[SWS_TtCanIf_00052] ⌈ If parameter `ControllerId` of `CanIf_TTGap()` has an invalid value and if development error detection is enabled (i.e. `CANIF_DEV_ERROR_DETECT` equals `ON`), then the function `CanIf_TTGap()` shall report development error code `CANIF_E_PARAM_CONTROLLER` to the `Det_ReportError` service of the DET module. ⌋()

8.6.5 CanIf_TTStartOfCycle

[SWS_TtCanIf_00092] ⌈

Service name:	CanIf_TTStartOfCycle	

Syntax:	`Std_ReturnType CanIf_TTStartOfCycle(`
	`uint8 ControllerId,`
	`uint8 CanIf_TTCycleCount`
	`)`	

Service ID[hex]:	0x5e	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	ControllerId CanIf_TTCycle Count	Abstracted CanIf ControllerId which is assigned to a CAN controller for which the start of cycle shall be reported. Cycle count value for the cycle that is started
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Function successful E_NOT_OK: Development error occurred
Description:	Reports the start of a basic cycle.	

Table 8.34: CanIf_TTStartOfCycle

⌋(*SRS_Can_01131*)

[SWS_TtCanIf_00053] ⌈ If parameter `ControllerId` of `CanIf_TTStartOfCycle()` has an invalid value and if development error

detection is enabled (i.e. CANIF_DEV_ERROR_DETECT equals ON), then the function CanIf_TTStartOfCycle() shall report development error code CANIF_E_PARAM_CONTROLLER to the Det_ReportError service of the DET module. ⌋()

8.6.6 CanIf_TTTimeDisc

[SWS_TtCanIf_00093] ⌈

Service name:	CanIf_TTTimeDisc	
Syntax:	Std_ReturnType CanIf_TTTimeDisc(uint8 ControllerId)	
Service ID[hex]:	0x5f	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN controller for which the time discontinuity shall be reported.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Function successful E_NOT_OK: Development error occurred
Description:	Reports a time discontinuity.	

Table 8.35: CanIf_TTTimeDisc

⌋(*SRS_Can_01131*)

[SWS_TtCanIf_00054] ⌈ If parameter ControllerId of CanIf_TTTimeDisc() has an invalid value and if development error detection is enabled (i.e. CANIF_DEV_ERROR_DETECT equals ON), then the function CanIf_TTTimeDisc() shall report development error code CANIF_E_PARAM_CONTROLLER to the Det_ReportError service of the DET module. ⌋()

8.6.7 CanIf_TTMasterStateChange

[SWS_TtCanIf_00094] ⌈

Service name:	CanIf_TTMasterStateChange
Syntax:	Std_ReturnType CanIf_TTMasterStateChange(uint8 ControllerId, CanIf_TTMasterStateType CanIf_TTMasterState)
Service ID[hex]:	0x60
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant

Parameters (in):	ControllerId	Abstracted CanIf ControllerId which is assigned to a CAN controller for which the master state change shall be reported.
	CanIf_TTMasterState	Master state including sync mode, master-slave mode and current ref trigger offset
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: Function successful E_NOT_OK: Development error occurred
Description:	Reports change of the master state between potential and current master.	

Table 8.36: CanIf_TTMasterStateChange

⌋(*SRS_Can_01131*)

[SWS_TtCanIf_00055] ⌈ If parameter `ControllerId` of `CanIf_TTMasterStateChange()` has an invalid value and if development error detection is enabled (i.e. `CANIF_DEV_ERROR_DETECT` equals `ON`), then the function `CanIf_TTMasterStateChange()` shall report development error code `CANIF_E_PARAM_CONTROLLER` to the `Det_ReportError` service of the DET module. ⌋()

8.7 Expected interfaces

8.7.1 Mandatory interfaces

Additional TTCAN specific mandatory interfaces

In this chapter defines all interfaces, required from other modules are listed.

[SWS_TtCanIf_00056] ⌈

API function	*Description*
Can_TTGetControllerTime	Gets the current values for the global, local and cycle time and the cycle count of the controller
Can_TTGetErrorLevel	Gets the error level. This includes the severity of the error level (S0-S3) and the minimum and maximum value of the message status count.
Can_TTGetMasterState	Gets the master state. The master state includes the sync mode (sync_off, synchronizing, in_gap, in_schedule) the master-slave mode (master_off, slave, backup_master, current_master) and the current value for ref trigger offset.
Can_TTGetNTUActual	Gets the actual value of NTU (network time unit). Together with the local oscillator period, the actual value of NTU can be derived from the actual value of TUR.
Can_TTGlobalTimePreset	Sets the value of "global time preset".
Can_TTSetEndOfGap	Signals the end of a gap.

Can_TTSetExtClockSyncCommand	Adjusts the NTU (network time unit) according to the value given by "NTU adjust". Together with the local oscillator period, "TUR adjust" can be derived from "NTU adjust".
Can_TTSetNextIsGap	Sets the "Next_is_Gap" bit.
Can_TTSetNTUAdjust	Sets the value of "NTU adjust". Together with the local oscillator period, "TUR adjust" can be derived from "NTU adjust".
Can_TTSetTimeCommand	Adjusts the global time at the beginning of the next basic cycle by the amount of "global time preset"
Dem_SetEventStatus	Called by SW-Cs or BSW modules to report monitor status information to the Dem. BSW modules calling Dem_SetEventStatus can safely ignore the return value.

Table 8.37: TtcanIf Mandatory Interfaces

⌋()

8.7.2 Optional Interfaces

Additional TTCAN specific optional interfaces

This chapter defines all interfaces which are required to fulfill an optional functionality of the module.

[SWS_TtCanIf_00057] ⌈

API function	*Description*
Can_TTAckTimeMark	Acknowledges the time mark interrupt by resetting the flag in the interrupt vector register.
Can_TTCancelTimeMark	Cancels the time mark.
Can_TTDisableTimeMarkIRQ	Disables the time mark interrupt.
Can_TTEnableTimeMarkIRQ	Enables the time mark interrupt.
Can_TTGetSyncQuality	Gets the synchronization quality.
Can_TTGetTimeMarkIRQStatus	Gets the IRQ status of the time mark.
Can_TTReceive	Reads received data from the controller by returning the pointer of the CanID, the DLC and the Data of the message in the requested HRH.
Can_TTSetTimeMark	Sets a new value for the time mark for the given time source.

Table 8.38: TtcanIf Optional Interfaces

⌋()

8.7.3 Configurable Interfaces

Additional TTCAN specific configurable interfaces

This chapter lists all interfaces where the target API service of any upper layer, which require one or more of these mentioned interfaces to be called has to be set up by static configuration of TtcanIf. The target function is usually a call-back function. The names of these kinds of interfaces are not fixed because they are configurable.

8.7.3.1 <User_TriggerTransmit>

[SWS_TtCanIf_00058] ⌈

Service name:	<User_TriggerTransmit>	
Syntax:	`Std_ReturnType <User_TriggerTransmit>( PduIdType TxPduId, PduInfoType* PduInfoPtr )`	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant for different PduIds. Non reentrant for the same PduId.	
Parameters (in):	TxPduId	ID of the SDU that is requested to be transmitted.
Parameters (inout):	PduInfoPtr	Contains a pointer to a buffer (SduDataPtr) to where the SDU data shall be copied, and the available buffer size in SduLengh. On return, the service will indicate the length of the copied SDU data in SduLength.
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: SDU has been copied and SduLength indicates the number of copied bytes. E_NOT_OK: No SDU data has been copied. PduInfoPtr must not be used since it may contain a NULL pointer or point to invalid data.
Description:	Within this API, the upper layer module (called module) shall check whether the available data fits into the buffer size reported by PduInfoPtr->SduLength. If it fits, it shall copy its data into the buffer provided by PduInfoPtr->SduDataPtr and update the length of the actual copied data in PduInfoPtr->SduLength. If not, it returns E_NOT_OK without changing PduInfoPtr.	

Table 8.39: <User_TriggerTransmit>

⌋()

When calling the `PduR`, this function has to be named `<User_TriggerTransmit>()`.

This API service of an upper layer BSW module `<User_>` (e.g. `PduR`) is called by `TtcanIf` to request from this upper layer BSW module that the PDU with index `TxPduId` has to be copied to the location in a temporary L-SDU buffer of `TtcanIf` to which this part of `PduInfoPtr` points.

[SWS_TtCanIf_00144] ⌈ If during `JLEF <User_TriggerTransmit>()` returns `E_NOT_OK`, `TtcanIf` shall not call `Can_Write()` afterwards (see Figure 9.1). Figure 9.1 shows only the case when `<User_TriggerTransmit>()` returns `E_OK`. ⌋()

Reason for [SWS_TtCanIf_00144]: It is possible that e.g. the PDU is not available in COM module. This may be due to a stopped PDU group in COM module. Caveats of `<User_TriggerTransmit>()`: This API service is called during the execution of the `TTCAN JLEF`.

9 Sequence diagrams

The following sequence diagrams show the interactions of `TtcanIf` additional to the CAN Interface.

9.1 Transmission with JobList (TriggerTransmit with decoupled buffer access)

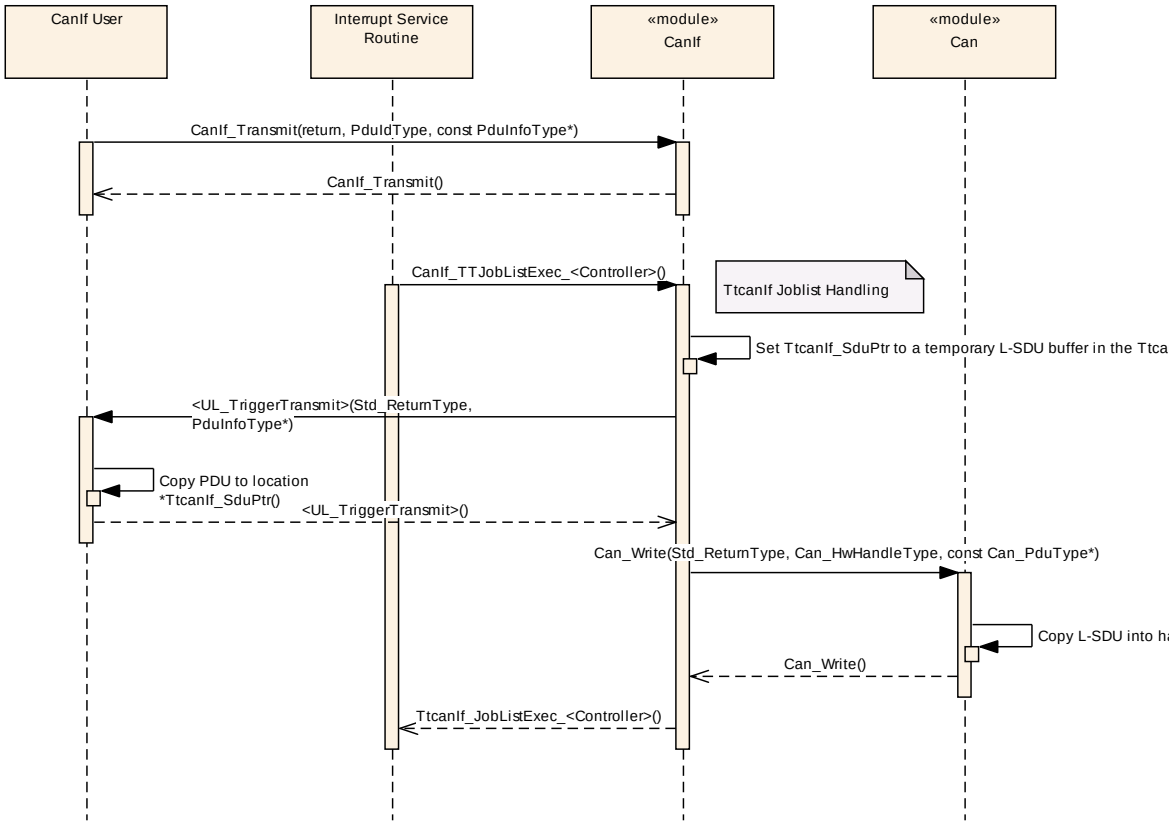

Figure 9.1: CAN Interface Time Triggered transmission with `Job List`

9.2 Reception with Joblist

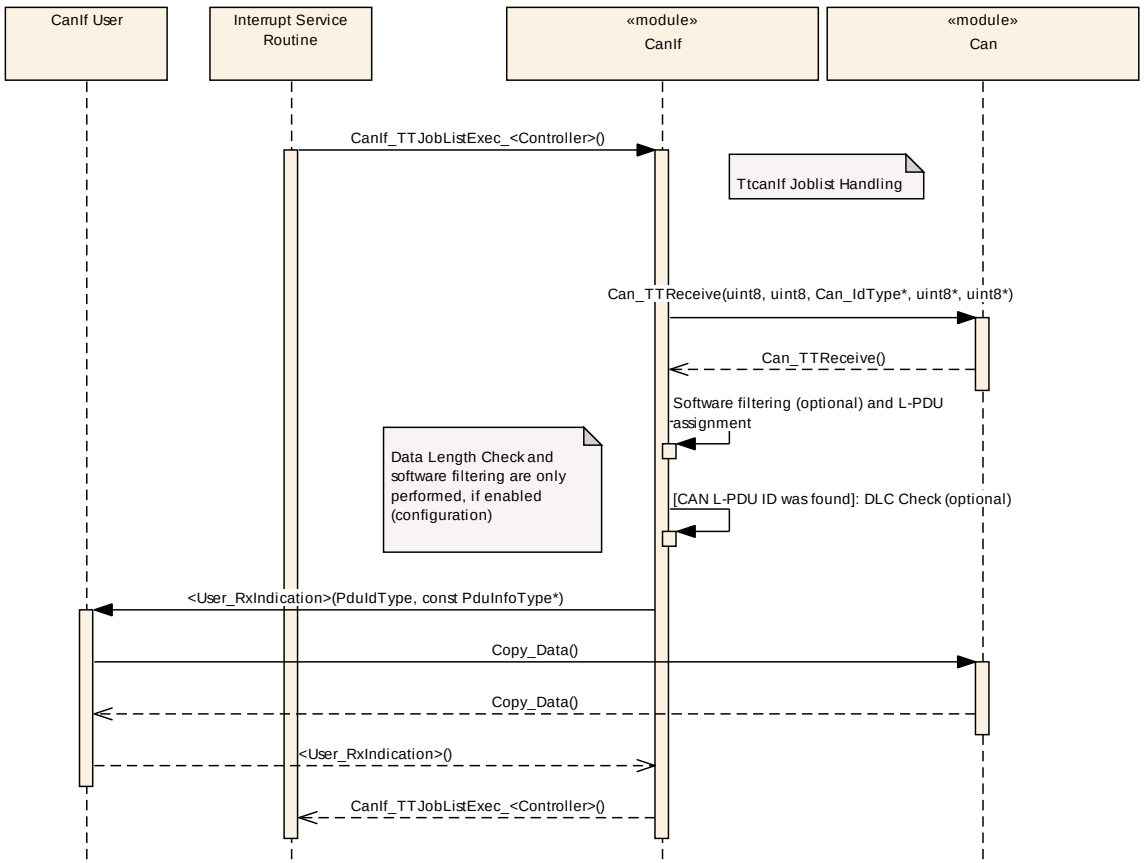

Figure 9.2: CAN Interface Time Triggered reception with `Job List`

9.3 Job List Execution Function

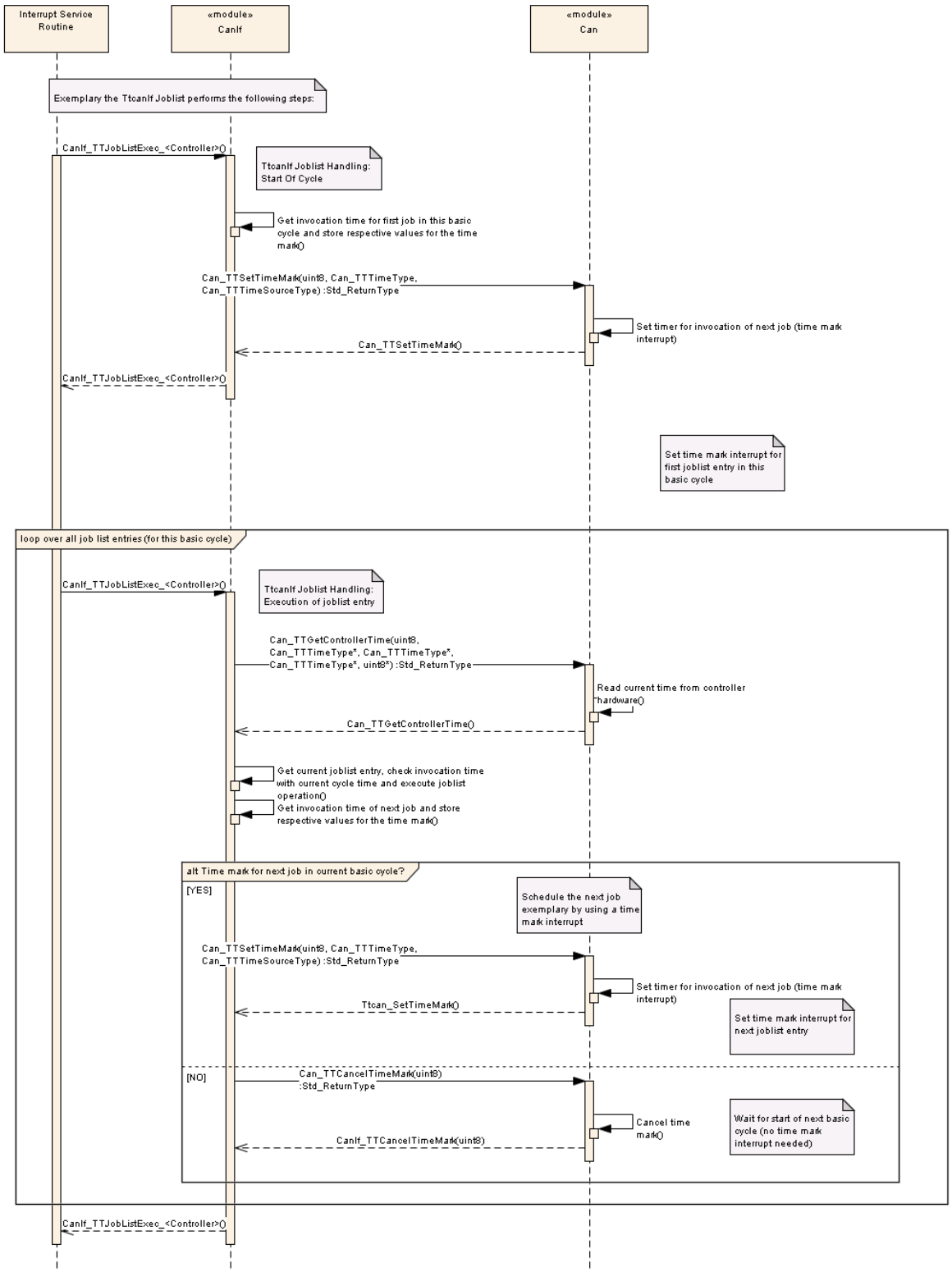

Figure 9.3: CAN Interface Time Triggered `Job List Execution Function` (`JLEF`)

10 Configuration specification

In general, this chapter defines configuration parameters and their clustering into containers. For general information about the definition of containers and parameters, refer to the [5, chapter 10.1 "Introduction to configuration specification" in SWS_BSWGeneral].

section 10.1 specifies the structure (containers) and the parameters of `TtcanIf`.

section 10.2 specifies published information of `TtcanIf`.

10.1 Containers and configuration parameters

Additional TTCAN specific configuration parameters

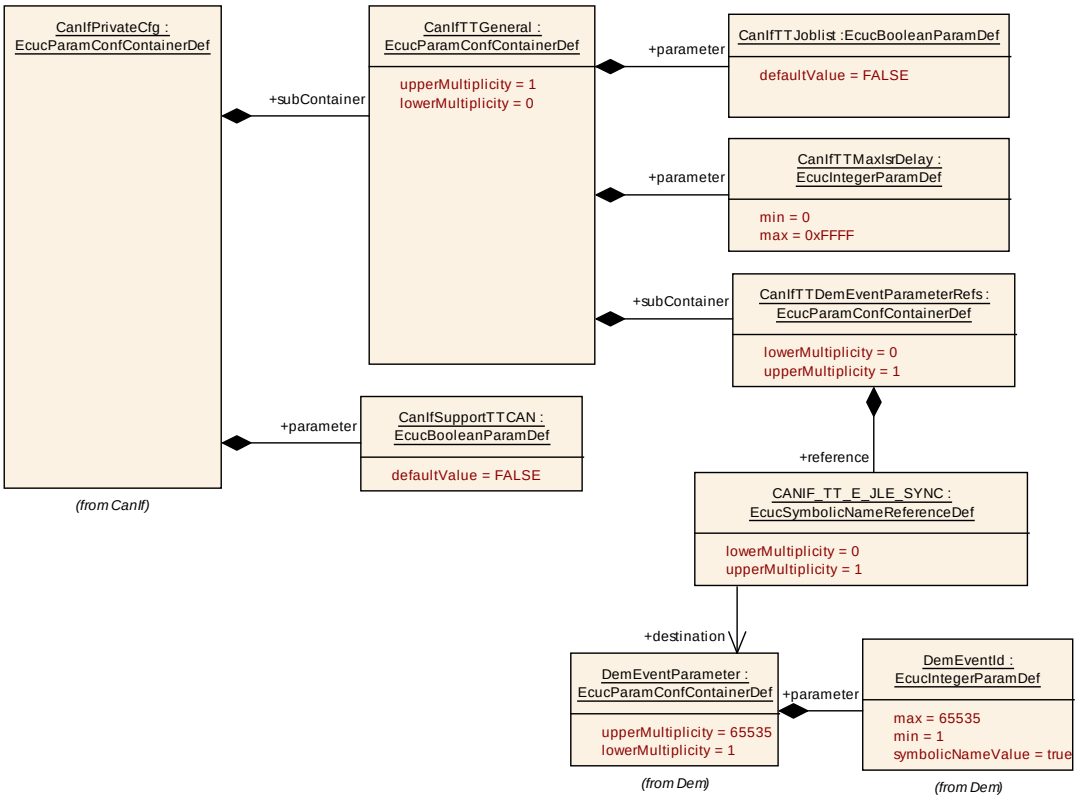

Figure 10.1: CAN Interface Time Triggered Private Configuration

The parameter `CanIfSupportTTCAN` is described in Specification of [2, CAN Interface SWS, ECUC_CanIf_00675].

10.1.1 CanIfTTGeneral

SWS Item	[ECUC_CanIf_00005]

Container Name	CanIfTTGeneral
Description	CanIfTTGeneral is specified in the SWS TTCAN Interface and defines if and in which way TTCAN is supported. This container is only included and valid if TTCAN is supported by the controller, enabled (see CanIfSupportTTCAN, ECUC_CanIf_00675), and used.
Configuration Parameters	

Name	CanIfTTJoblist [ECUC_CanIf_00126]		
Parent Container	CanIfTTGeneral		
Description	Defines whether TTCAN is processed via a joblist. TRUE: Joblist is used. FALSE: No joblist is used. This parameter is only configurable if TTCAN is enabled by parameter CanIfSupportTTCAN.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default Value	false		
Post-Build Variant Value	false		
Value Configuration Class	**Pre-compile time**	X	All Variants
	Link time	–	
	Post-build time	–	
Scope / Dependency	scope: local dependency: CanIfSupportTTCAN		

Name	CanIfTTMaxIsrDelay [ECUC_CanIf_00127]		
Parent Container	CanIfTTGeneral		
Description	Defines the maximum delay for the execution of the joblist execution function JLEF. This parameter is only configurable if a joblist is enabled by parameter CanIfTTJobList.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default Value			
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local dependency: CanIfTTJobList		

Included Containers		
Container Name	**Multiplicity**	**Scope / Dependency**
CanIfTTDemEvent ParameterRefs	0..1	Container for the references to DemEventParameter elements which shall be invoked using the API Dem_SetEventStatus in case the corresponding error occurs. The EventId is taken from the referenced DemEventParameter's DemEventId symbolic value. The standardized errors are provided in this container and can be extended by vendor-specific error references.

10.1.2 CanIfTTDemEventParameterRefs

SWS Item	[ECUC_CanIf_00835]
Container Name	CanIfTTDemEventParameterRefs
Description	Container for the references to DemEventParameter elements which shall be invoked using the API Dem_SetEventStatus in case the corresponding error occurs. The EventId is taken from the referenced DemEventParameter's DemEventId symbolic value. The standardized errors are provided in this container and can be extended by vendor-specific error references.
Configuration Parameters	

Name	CANIF_TT_E_JLE_SYNC [ECUC_CanIf_00836]		
Parent Container	CanIfTTDemEventParameterRefs		
Description	Reference to configured DEM event to report that the JLEF lost synchronization to the local time of the TTCAN controller.		
Multiplicity	0..1		
Type	Symbolic name reference to DemEventParameter		
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local dependency: Dem		

No Included Containers

10.1.3 CanIfTTTxFrameTriggering

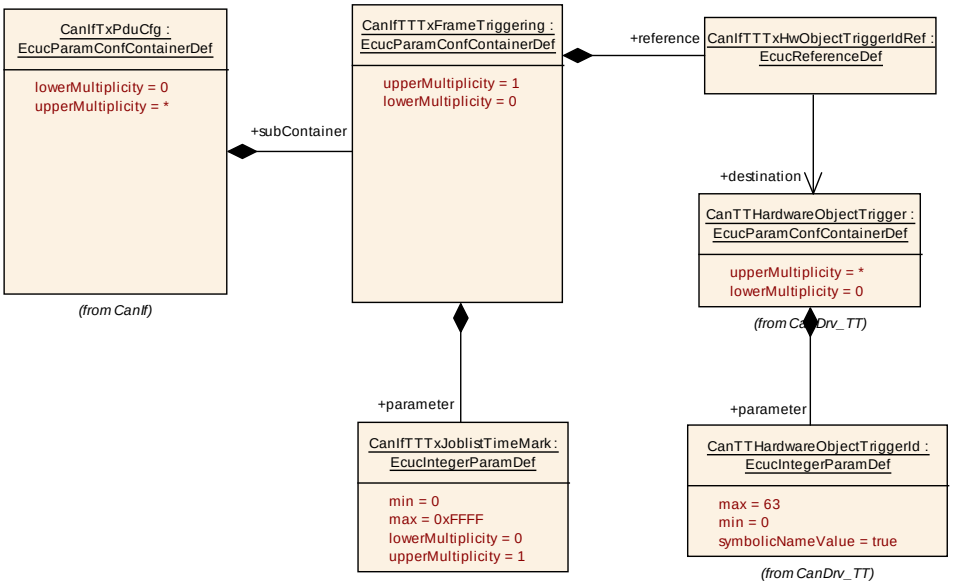

Figure 10.2: CAN Interface Time Triggered Transmit PDU Configuration

SWS Item	[ECUC_CanIf_00142]
Container Name	CanIfTTTxFrameTriggering
Description	CanIfTTTxFrameTriggering is specified in the SWS TTCAN Interface and defines Frame trigger for TTCAN transmission. This container is only included and valid if TTCAN is supported by the controller, enabled (see CanIfSupportTTCAN, ECUC_CanIf_00675), and a joblist is used.
Configuration Parameters	

Name	CanIfTTTxJoblistTimeMark [ECUC_CanIf_00132]		
Parent Container	CanIfTTTxFrameTriggering		
Description	Defines the point in time, when the joblist execution funciton (JLEF) shall be called for the referenced tx frame trigger. Value is given in cycle time. This parameter is only configurable if a joblist is enabled by parameter CanIfTTJobList.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default Value			
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD

Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local dependency: CanIfTTJoblist		

Name	CanIfTTTxHwObjectTriggerIdRef [ECUC_CanIf_00128]		
Parent Container	CanIfTTTxFrameTriggering		
Description	This parameter refers to a particular TTCAN hardware transmit object Trigger of a hardware object in the TTCAN Driver Module, which is referred via plain CAN parameter CANIF_HTH_HANDLETYPE_REF. This parameter is only configurable if a joblist is enabled by parameter CanIfTTJobList.		
Multiplicity	1		
Type	Reference to CanTTHardwareObjectTrigger		
Post-Build Variant Value	true		
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local dependency: CanIfTTJoblist		

No Included Containers

10.1.4 CanIfTTRxFrameTriggering

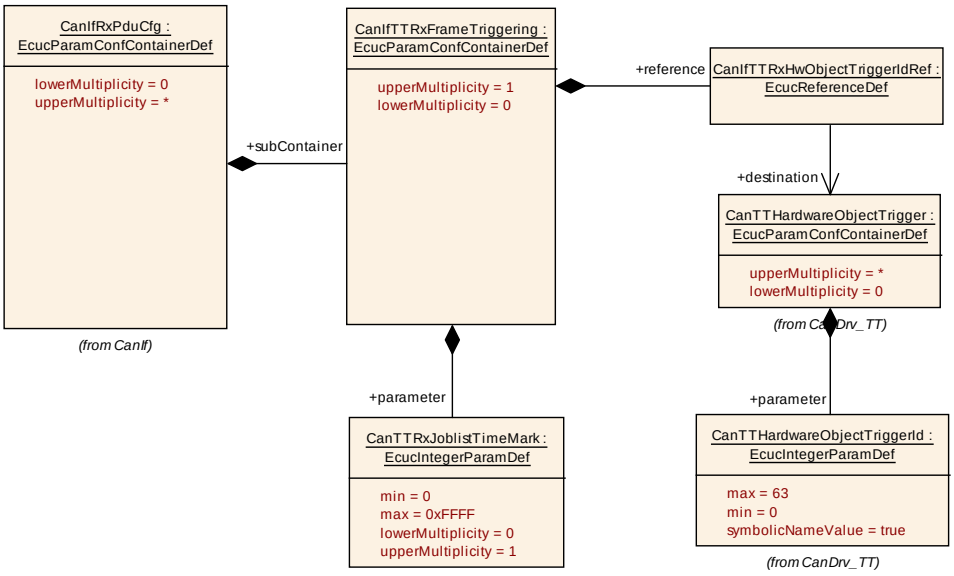

Figure 10.3: CAN Interface Time Triggered Receive PDU Configuration

SWS Item	[ECUC_CanIf_00003]
Container Name	CanIfTTRxFrameTriggering
Description	CanIfTTRxFrameTriggering is specified in the SWS TTCAN Interface and defines Frame trigger for TTCAN reception. This container is only included and valid if TTCAN is supported by the controller, enabled (see CanIfSupportTTCAN, ECUC_CanIf_00675), and a joblist is used for reception.
Configuration Parameters	

Name	CanTTRxJoblistTimeMark [ECUC_CanIf_00136]		
Parent Container	CanIfTTRxFrameTriggering		
Description	Defines the point in time, when the joblist execution funciton (JLEF) shall be called for the referenced rx trigger. Value is given in cycle time. This parameter is only configurable if a joblist is enabled by parameter CanIfTTJobList.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default Value			
Post-Build Variant Multiplicity	true		
Post-Build Variant Value	true		
Multiplicity Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local dependency: CanIfTTJoblist		

Name	CanIfTTRxHwObjectTriggerIdRef [ECUC_CanIf_00133]
Parent Container	CanIfTTRxFrameTriggering
Description	This parameter refers to a particular TTCAN hardware receive object Trigger of a hardware object in the TTCAN Driver Module, which is referred via plain CAN parameter CANIF_HRH_HANDLETYPE_REF. This parameter is only configurable if a joblist is enabled by parameter CanIfTTJobList.
Multiplicity	1
Type	Reference to CanTTHardwareObjectTrigger
Post-Build Variant Value	true

Value Configuration Class	**Pre-compile time**	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency	scope: local dependency: CanIfTTJoblist		

No Included Containers

10.2 Published information

For details refer to the [5, chapter 10.1 "Published Information" in SWS_BSWGeneral]

A Not applicable requirements

[SWS_TtCanIf_99999] ⌈ These requirements are not applicable to this specification. ⌋ *()*