Document Title	Requirements on CAN
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	1

Document Status	published
Part of AUTOSAR Standard	Classic Platform
Part of Standard Release	R24-11

Document Change History

Date	Release	Changed by	Description
2024-11-27	R24-11	AUTOSAR Release Management	• Minor corrections / clarifications / editorial changes • Modified from draf to valid [SRS_Can_02001], [SRS_Can_02002], [SRS_Can_02003] • Introduced Deterministic Communication with TSN : Obsolete requirements [SRS_Can_01002], [SRS_Can_01003], [SRS_Can_01111] Added Requirements [SRS_Can_02004], [SRS_Can_02005], [SRS_Can_02006], [SRS_Can_02007]
2023-11-23	R23-11	AUTOSAR Release Management	• Minor corrections / clarifications / editorial changes
2022-11-24	R22-11	AUTOSAR Release Management	• CanXL requirements were added • Minor corrections / clarifications / editorial changes
2020-11-30	R20-11	AUTOSAR Release Management	• Editorial changes
2019-11-28	R19-11	AUTOSAR Release Management	• Bus-independent solution regarding channel states upon initialization • Changed Document Status from Final to published

▽

△

2018-10-31	4.4.0	AUTOSAR Release Management	• Added requirements for BusMirroring • Removed half-duplex mode from CanTp
2016-12-08	4.3.1	AUTOSAR Release Management	• Editorial changes
2016-11-30	4.3.0	AUTOSAR Release Management	• Added method to obtain error active/passive state of a CAN
2014-10-31	4.2.1	AUTOSAR Release Management	• Added requirements for CAN FD support • Removed requirements for transmit cancellation
2014-03-31	4.1.3	AUTOSAR Release Management	• Revised DLC checks depending on padding configuration
2013-10-31	4.1.2	AUTOSAR Release Management	• Corrected requirement for: "Do not send WUF as First Message on the Bus after BusOff" • Editorial changes
2013-10-15	4.1.1	AUTOSAR Administration	• Support for 29bit Mixed Addressing • Wakeup by bus callback shall be synchronous or asynchronous depending on the hardware • Advanced transmit buffer handling
2011-12-22	4.0.3	AUTOSAR Administration	• Added high level requirements for partial networking • Added improvement of transmit buffer handling • Added full duplex support
2011-04-15	4.0.2	AUTOSAR Administration	• BSW01017 requirement for CAN polling/interrupt mode removed
2010-09-30	3.1.5	AUTOSAR Administration	• Additional requirements for transport layer CAN • Requirement for remote frame support added • Legal disclaimer revised
2008-08-13	3.1.1	AUTOSAR Administration	• Legal disclaimer revised

▽

△

2007-07-24	2.1.16	AUTOSAR Administration	• "Advice for users" revised • "Revision Information" added
2007-01-24	2.1.15	AUTOSAR Administration	• PDF file corrections made
2006-11-28	2.1	AUTOSAR Administration	• Architecture design change: CAN Transceiver Driver is now layered below CAN Interface • Extended 11/29 bit Identifier support in CAN Interface • Added N_SA in [SRS_Can_01069] and [SRS_Can_01074] • Legal disclaimer revised
2006-05-16	2.0	AUTOSAR Administration	• CAN Driver, CAN Interface • Optimized timing behavior for transmission (multiplexed transmission, priority based transmission, transmission cancellation) • Support of Standard and Extended CAN Identifiers on one network • CAN Transport Layer • Multiple connections mechanism, • Support of ISO-15765-4, • Support of Connection specific time out values • Support of different addressing modes in parallel • CAN Transceiver Driver • Requirements for CAN Transceiver Driver added
2005-05-31	1.0	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.

Contents

1 Scope of Document

This document specifies the requirements for the following Basic Software Modules (module names in brackets):

- CAN Driver (Can[1])
- CAN Interface (CanIf[2])
- CAN State Manager (CanSM[3])
- CAN Transport Layer (CanTp[4])
- CAN Bus Transceiver Driver (CanTrcv[5])

2 How to read this document

Each requirement has its unique identifier starting with the prefix "BSW" (for "Basic Software"). For any review annotations, remarks or questions, please refer to this unique ID rather than chapter or page numbers!

2.1 Document Conventions

The representation of requirements in AUTOSAR documents follows the table specified in [TPS_STDT_00078], see Standardization Template, chapter Support for Traceability ([6]).

The verbal forms for the expression of obligation specified in [TPS_STDT_00053] shall be used to indicate requirements, see Standardization Template, chapter Support for Traceability ([6]).

2.2 Requirements Structure

Each module specific chapter contains a short functional description of the Basic Software Module. Requirements of the same kind within each chapter are grouped under the following headlines (where applicable):

Functional Requirements:

- Configuration (which elements of the module need to be configurable)
- Initialization
- Normal Operation
- Shutdown Operation
- Fault Operation
- ...

Non-Functional Requirements:

- Timing Requirements
- Resource Usage
- Usability
- Output for other WPs (e.g. Description Templates, Tooling,...)
- ...

3 Acronyms and Abbreviations

Acronym:	Description:
CAN Communication Matrix	Describes the complete CAN network: • Participating nodes • Definition of all CAN PDUs (Identifier, DLC) • Source and Sinks for PDUs Format is defined in other AUTOSAR workpackage
Physical Channel	A physical channel represents an interface to the CAN Network. Different physical channels of the CAN Hardware Unit may access different networks.
L-PDU	CAN (Data Link Layer) Protocol Data Unit. Consists of Identifier, DLC and Data (L-SDU).
L-SDU	CAN (Data Link Layer) Service Data Unit. Data that is transported inside the L-PDU.
Hardware Object	A Hardware Object is defined as message buffer inside the CAN RAM of the CAN Hardware Unit. Also often called Message Object
Hardware Object Handle	The hardware object handle (HOH) is defined and provided by the CAN Driver. Typically each HOH represents a hardware object. The HOH is used as parameter by the CAN Interface Layer for transmit and read requests to the CAN Driver.
L-PDU Handle	The L-PDU handle is defined and placed inside the CAN Interface Layer. Typically each handle represents a L-PDU or a range of L-PDUs, and is a constant structure with information for Tx/Rx processing.
CAN Controller	A CAN controller serves exactly one physical channel. See Figure "Typical CAN HW Unit" in CAN Interface SWS.
CAN Hardware Unit	A CAN hardware unit may consist of one or multiple CAN controllers of the same type and one or multiple CAN RAM areas. The CAN hardware unit is either on-chip, or an external device. The CAN hardware unit is represented by one CAN Driver. See Figure "Typical CAN HW Unit" in CAN Interface SWS.
Multiplexed Transmission	Usage of three TX HW objects, which are represented as one transmit entity (Hardware Object Handle) to the upper layer. Used for Outer Priority Inversion avoidance
Inner Priority Inversion	Transmission of a high-priority L-PDU is prevented by the presence of a pending low-priority L-PDU in the same physical channel.
Outer Priority Inversion	Occurs when a time gap is between two consecutive TX L-PDU transmissions. In this case a lower priority L-PDU from another node can prevent sending the next L-PDU because the higher priority L-PDU can't participate in the running bus arbitration because it comes too late.
Bus	A bus represents a CAN or LIN network. A bus has a given physical behavior (e.g. CAN low-speed or high-speed). A bus may support wakeup via bus or is "always on".
N-PDU	Network Protocol Data Unit of the CAN Transport Layer
N-SDU	Service Data Unit of the CAN Transport Layer. Data that is transported inside the N-PDU.
static configuration	Configuration, that is not changeable during runtime. This means that a configuration is typically done once during startup phase of the ECU. This concern is independent from the possibilities to introduce the configuration parameters into the ECU itself: Pre-Compile-Time, Link-Time or Post-Build-Time
STmin	Separation Time min
BS	Block Size

HTH	CAN hardware transmit handle

Table 3.1: Acronyms and Abbreviations

4 Functional Overview

The CAN bus transceiver driver is responsible to handle the CAN transceivers on an ECU according to the expected state of the bus specific NM in relation to the current state of the whole ECU.

The transceiver is a hardware device, which mainly transforms the logical on/off signal values of the μC ports to the bus compliant electrical levels, currents and timings. Within an automotive environment there are mainly three different CAN physics used. These physics are ISO11898 for high-speed CAN (up to 1Mbd), ISO11519 for low-speed CAN (up to 125kBd). Both are regarded in AUTOSAR, whereas SAE J2411 for single-wire CAN is not. CAN FD utilizes the same CAN physic as it is used for high-speed CAN but provide faster transmission rates.

In addition, the transceivers are often able to detect electrical malfunctions like wiring issues, ground offsets or transmission of too long dominant signals. Depending on the interface they flag the detected error summarized by a single port pin or very detailed via SPI.

Some transceivers also support power supply control and wakeup via the bus. A lot of different wakeup/sleep and power supply concepts are available on the market with focus to best-cost optimized solution for a given task.
Latest developments are so called SystemBasisChips (SBC) where not only the CAN and/or LIN transceivers but also power-supply control and advanced watchdogs are implemented in one housing and are controlled via one interface (typically an SPI).

A typical CAN transceiver is the TJA1054 for a low-speed CAN bus. The same state transition model is also used in TJA1041 (high-speed CAN with support for wakeup via CAN) and could be transferred also to a lot of other products on the market.

Transceiver Wakeup Reason
The transceiver driver is able to store the local view on who has requested the wakeup: bus or software.

Bus: The bus has caused the wakeup.

Internally: The wakeup has been caused by a software request to the driver.

Sleep: The transceiver is in operation mode sleep and no wakeup has been occurred.

5 Requirements Tracing

Requirement	Description	Satisfied by
[RS_BRF_01000]	AUTOSAR architecture shall organize the BSW in a hardware independent and a hardware dependent layer	[SRS_Can_01001] [SRS_Can_01121]
[RS_BRF_01008]	AUTOSAR shall organize the hardware dependent layer in a microcontroller independent and a microcontroller dependent layer	[SRS_Can_01121]
[RS_BRF_01016]	AUTOSAR shall provide a modular design inside software layers	[SRS_Can_01121]
[RS_BRF_01056]	AUTOSAR BSW modules shall provide standardized interfaces	[SRS_Can_01142]
[RS_BRF_01064]	AUTOSAR BSW shall provide callback functions in order to access upper layer modules	[SRS_Can_01014] [SRS_Can_01045] [SRS_Can_01106] [SRS_Can_01138]
[RS_BRF_01088]	AUTOSAR shall offer interfaces which allow to express high level application communication needs	[SRS_Can_01154]
[RS_BRF_01096]	AUTOSAR shall support start-up and shutdown of ECUs	[SRS_Can_01108]
[RS_BRF_01104]	AUTOSAR shall support sleep and wake-up of ECUs and buses	[SRS_Can_01151] [SRS_Can_01156]
[RS_BRF_01136]	AUTOSAR shall support variants of configured BSW data resolved after system start-up	[SRS_Can_01021] [SRS_Can_01022] [SRS_Can_01023] [SRS_Can_01041] [SRS_Can_01090] [SRS_Can_01139] [SRS_Can_01155]
[RS_BRF_01152]	AUTOSAR shall support limited dynamic reconfiguration	[SRS_Can_01042]
[RS_BRF_01184]	AUTOSAR shall support different methods of degradation	[SRS_Can_01154]
[RS_BRF_01408]	AUTOSAR shall provide a service layer that is accessible from each basic software layer	[SRS_Can_01055]
[RS_BRF_01544]	AUTOSAR communication shall define transmission and reception of communication data	[SRS_Can_01003] [SRS_Can_01007] [SRS_Can_01008] [SRS_Can_01009] [SRS_Can_01011] [SRS_Can_01045] [SRS_Can_01049] [SRS_Can_01051] [SRS_Can_01109] [SRS_Can_01129] [SRS_Can_01131] [SRS_Can_02005]
[RS_BRF_01552]	AUTOSAR communication shall separate bus independent functionality from bus dependent functionality	[SRS_Can_01001] [SRS_Can_01034]
[RS_BRF_01600]	AUTOSAR communication shall support time-out handling	[SRS_Can_01081] [SRS_Can_01082] [SRS_Can_01143] [SRS_Can_01144] [SRS_Can_01146]
[RS_BRF_01608]	AUTOSAR communication shall support to filter signals	[SRS_Can_01004]
[RS_BRF_01632]	AUTOSAR communication shall support data consistency of groups of signals	[SRS_Can_01059] [SRS_Can_01114]

△

Requirement	Description	Satisfied by
[RS_BRF_01664]	AUTOSAR communication shall support a state management of buses	[SRS_Can_01027] [SRS_Can_01028] [SRS_Can_01029] [SRS_Can_01032] [SRS_Can_01054] [SRS_Can_01055] [SRS_Can_01060] [SRS_Can_01107] [SRS_Can_01115] [SRS_Can_01122] [SRS_Can_01136] [SRS_Can_01143] [SRS_Can_01144] [SRS_Can_01146] [SRS_Can_01156] [SRS_Can_01157]
[RS_BRF_01680]	AUTOSAR communication shall support mechanism to keep a bus awake, and to be kept awake by a bus	[SRS_Can_01006] [SRS_Can_01013] [SRS_Can_01032] [SRS_Can_01106] [SRS_Can_01107] [SRS_Can_01115] [SRS_Can_01136] [SRS_Can_01138] [SRS_Can_01151] [SRS_Can_01153] [SRS_Can_01156] [SRS_Can_01157]
[RS_BRF_01704]	AUTOSAR communication shall support the CAN communication bus	[SRS_Can_01002] [SRS_Can_01003] [SRS_Can_01004] [SRS_Can_01005] [SRS_Can_01006] [SRS_Can_01007] [SRS_Can_01008] [SRS_Can_01009] [SRS_Can_01011] [SRS_Can_01013] [SRS_Can_01015] [SRS_Can_01016] [SRS_Can_01018] [SRS_Can_01019] [SRS_Can_01020] [SRS_Can_01021] [SRS_Can_01022] [SRS_Can_01023] [SRS_Can_01027] [SRS_Can_01028] [SRS_Can_01029] [SRS_Can_01032] [SRS_Can_01033] [SRS_Can_01034] [SRS_Can_01035] [SRS_Can_01036] [SRS_Can_01037] [SRS_Can_01038] [SRS_Can_01039] [SRS_Can_01041] [SRS_Can_01042] [SRS_Can_01043] [SRS_Can_01045] [SRS_Can_01049] [SRS_Can_01051] [SRS_Can_01053] [SRS_Can_01054] [SRS_Can_01055] [SRS_Can_01058] [SRS_Can_01059] [SRS_Can_01060] [SRS_Can_01061] [SRS_Can_01062] [SRS_Can_01066] [SRS_Can_01068] [SRS_Can_01069] [SRS_Can_01071] [SRS_Can_01073] [SRS_Can_01074] [SRS_Can_01075] [SRS_Can_01076] [SRS_Can_01078] [SRS_Can_01079] [SRS_Can_01081] [SRS_Can_01082] [SRS_Can_01090] [SRS_Can_01091] [SRS_Can_01096] [SRS_Can_01097] [SRS_Can_01098] [SRS_Can_01099] [SRS_Can_01100] [SRS_Can_01101] [SRS_Can_01103] [SRS_Can_01106] [SRS_Can_01107] [SRS_Can_01108] [SRS_Can_01109] [SRS_Can_01110] [SRS_Can_01114] [SRS_Can_01115] [SRS_Can_01122] [SRS_Can_01125] [SRS_Can_01126] [SRS_Can_01129] [SRS_Can_01130] [SRS_Can_01131] [SRS_Can_01132] [SRS_Can_01134] [SRS_Can_01135] [SRS_Can_01136] [SRS_Can_01138] [SRS_Can_01139] [SRS_Can_01140] [SRS_Can_01141] [SRS_Can_01143] [SRS_Can_01144] [SRS_Can_01145] [SRS_Can_01146] [SRS_Can_01147] [SRS_Can_01149] [SRS_Can_01151] [SRS_Can_01153] [SRS_Can_01154] [SRS_Can_01155] [SRS_Can_01156] ▽

▽

△

Requirement	Description	Satisfied by
		△ [SRS_Can_01157] [SRS_Can_01158] [SRS_Can_01159] [SRS_Can_02002] [SRS_Can_02004] [SRS_Can_02005] [SRS_Can_02006]
[RS_BRF_01712]	AUTOSAR communication shall support the adaptable speed offered by CAN FD	[SRS_Can_01073] [SRS_Can_01095] [SRS_Can_01160] [SRS_Can_01161] [SRS_Can_01162] [SRS_Can_01163] [SRS_Can_02001] [SRS_Can_02003]
[RS_BRF_01720]	AUTOSAR communication shall support the standardized transport protocol for Diagnostics over CAN	[SRS_Can_01065] [SRS_Can_01066] [SRS_Can_01068] [SRS_Can_01069] [SRS_Can_01071] [SRS_Can_01073] [SRS_Can_01074] [SRS_Can_01075] [SRS_Can_01076] [SRS_Can_01078] [SRS_Can_01079] [SRS_Can_01081] [SRS_Can_01082] [SRS_Can_01086] [SRS_Can_01111] [SRS_Can_01112] [SRS_Can_01116] [SRS_Can_01148] [SRS_Can_01149] [SRS_Can_02007]
[RS_BRF_01728]	AUTOSAR communication shall support J1939 transport protocol	[SRS_Can_01159]
[RS_BRF_01736]	AUTOSAR communication shall support dynamic allocation of addresses as requested by J1939 network management	[SRS_Can_01159]
[RS_BRF_02168]	AUTOSAR diagnostics shall provide a central classification and handling of abnormal operative conditions	[SRS_Can_01082]

Table 5.1: Requirements Tracing

6 Requirements Specification

6.1 Remarks to the CAN Bus Transceiver Driver

CAN bus transceivers are very different in their behavior and supported features. The range starts with very simple CAN transceivers, which are "always on", includes transceivers with support for advanced limp home handling and error detection and ends with so called system basis chips (SBC) which contain internally multiple CAN bus transceivers, watchdog, voltage regulators and more.
The size of transceiver data sheets is from few pages to more than 80 pages and the additional application notes for the devices are nearly countless.

The target of this document is to specify interfaces and behavior, which is applicable to most current and future CAN bus transceivers on the market for nearly all use cases. If it could be reached that at least the "user" of the bus transceiver functionality, typically the AUTOSAR NM and the AUTOSAR Communication Manager, are bus independent and therefore reusable, will be great.

It will not be possible to cover all possible combinations of bus transceivers with all conceivable power concepts within one AUTOSAR implementation.

6.1.1 Explicitly uncovered CAN Bus Transceiver functionality

Some CAN bus transceivers offer additional functionality to improve e.g. ECU self test or enhanced error detection capability for diagnostics.

ECU self test and enhanced error detection are not defined within AUTOSAR and requiring such functionality in general will lock out most currently used (and cheap) transceiver devices. Therefore features like "ground shift detection", "selective wakeup", "slope control" and others are not supported within this requirement. A general and "open" API like IOControl() is not applicable (and accepted) within AUTOSAR due to portability and reuse.

6.1.2 System Basis Chip and CAN Bus Transceiver Driver

A system basis chip (SBC) contains beside the CAN bus transceivers additional hardware related to power control and safety (e.g. multiple voltage regulators and a watchdog) and even more features (e.g. persistent memory).

In the AUTOSAR concept, a separate manager/driver/handler (in AUTOSAR called: Interface) is responsible for each identified hardware device. Therefore additional manager/driver/handler covers the functionality inside a SBC beside the bus transceiver driver (e.g. Watchdog Manager, non-volatile memory manager, power control driver, ...). Due to the shared communication access and the (security-related) restrictions within this communication, independent handling of each SBC-sub-functionality will not be possible.
This will lead to the situation that either a SBC could not be used within an AUTOSAR compliant ECU or (the better solution) a specialized manager/driver/handler for the SBC functionality with all APIs of each single domain has to be used.

6.2 Functional Requirements

6.2.1 CAN Driver

The CAN Driver offers uniform interfaces for the above user of this layer, the CAN Interface. The CAN Driver hides the hardware specific properties of the related CAN Controller as far as possible and reasonable.

For a detailed functional description and interface definition see CAN Driver Specification [Can[1]].

6.2.1.1 Configuration

[SRS_Can_01036] The Can Driver shall support Standard Identifier and Extended Identifier

Upstream requirements: RS_BRF_01704

⌈

Description:	The CAN driver shall be able to operate with both standard and extended CAN Identifiers on one CAN Controller if supported by CAN Hardware. Each hardware object shall be statically and individually configurable for one of the both identifier types if supported by CAN Hardware. All L-PDUs sent and received over that CAN controller shall be conform this configuration. The CAN Driver shall support reception and transmission of L-PDUs with Standard and Extended ID, including both at the same time on one Hardware Object. The configuration parameters shall be allowed to be of types Pre-Compile-Time, Link-Time or Post-Build-Time
Rationale:	CAN Standard Coverage

⌋

△

Use Case:	CAN Standard allows Standard and Extended Identifier. Different projects might require the usage of Extended CAN IDs in addition to Standard CAN IDs due to the lack of remaining StandardCAN IDs.
Dependencies:	[SRS_Can_01016]
Supporting Material:	–

⌋

[SRS_Can_01037] The CAN driver shall allow the static configuration of the hardware reception filter

Upstream requirements: RS_BRF_01704

⌈

Description:	HW supported filtering of receive L-PDUs shall be configurable. The configuration shall be done during initialization phase. Reconfiguration during normal operation shall only be possible in STOPPED mode. It shall be allowed for the configuration parameters to be of types Pre-Compile, Link-Time or Post-Build
Rationale:	Coverage of hardware capabilities
Use Case:	CAN controller allow filtering of messages inside hardware. That reduces the software load caused by messages not relevant for the ECU.
Dependencies:	[SRS_Can_01018]
Supporting Material:	–

⌋

[SRS_Can_01038] The bit timing of each CAN Controller shall be configurable

Upstream requirements: RS_BRF_01704

⌈

Description:	The bit timing and thus the Baud Rate of each CAN controller served by the CAN Driver shall be configurable The following list describes typical attributes: • Propagation delay • Tseg1 • Tseg2 • Samples/bit • SJW The configuration parameters shall be allowed to be of types Pre-Compile-Time, Link-Time or Post-Build-Time

▽

△

Rationale:	CAN Standards coverage, coverage of hardware capabilities
Use Case:	CAN Standard doesn't specify one baud rate -> baud rate is project specific. Possible configuration of the timing parameters is hardware dependent
Dependencies:	[SRS_Can_01139]
Supporting Material:	–

⌋

[SRS_Can_01039] Hardware Object Handles shall be provided for the CAN Interface in the static configuration file.

Upstream requirements: RS_BRF_01704

⌈

Description:	All available hardware object handles shall be defined in the ECU configuration description. The syntax of the public part shall be standardized, because that is the configuration interface to the CAN Interface The configuration parameters shall be allowed to be of types Pre-Compile-Time, Link-Time or Post-Build-Time
Rationale:	Coverage of hardware capabilities, configuration interface to CAN Interface
Use Case:	For an optimized co-operation of software and hardware filtering and optimized usage of underlying hardware the CAN Interface needs to know the available hardware resources and their configuration.
Dependencies:	[SRS_Can_01016]
Supporting Material:	–

⌋

[SRS_Can_01058] shall be configurable whether Multiplex Transmission is used

Upstream requirements: RS_BRF_01704

⌈

Description:	The Multiplexed Transmission feature shall be Pre-Compile-Time configurable. This feature shall only be supported if the underlying CAN Controller supports Multiplexed Transmission
Rationale:	–
Use Case:	Outer priority inversion can be avoided
Dependencies:	[SRS_Can_01134]
Supporting Material:	–

⌋

[SRS_Can_01062] Each event for each CAN Controller shall be configurable to be detected by polling or by an interrupt

Upstream requirements: RS_BRF_01704

⌈

Description:	Each possible event of each CAN Controller shall be Pre-Compile-Time configurable to be in one of the following two modes Polling: The CAN Driver represents at least one periodically called task. It polls the CAN Controller. The appropriate notifications are called based upon the events that occurred. It is optional for the CAN Driver to support multiple poll cycles. The CAN interrupt for the appropriate event is disabled in that mode. Interrupt driven: The CAN Controller notifies the CAN Driver of detected HW events by way of an interrupt. CAN Hardware Unit implementations may differ in regards to which events may be reported by interrupts or can only be polled -> The configuration for polling or interrupt shall be done inside the driver
Rationale:	Coverage of hardware capabilities
Use Case:	Polling mode is required when a deterministic timing behavior (response time) is needed. For example for motor management systems.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01135] It shall be possible to configure one or several TX Hardware Objects

Upstream requirements: RS_BRF_01704

⌈

Description:	It shall be possible to configure one or several TX Hardware Objects, where each Hardware Object is represented by it's own Hardware Object Handle. (Not to be mixed-up with multiplexed transmission) The selection of the TX Hardware Object is done by the caller of the transmit request service, with a parameter that identifies the Hardware Object Handle This requires that the hardware allows configuration of several TX Hardware Objects. The configuration shall be allowed to be of types Pre-Compile, Link-Time or Post-Build
Rationale:	Basic functionality
Use Case:	Support of typical CAN Controller capabilities: Configuration of several Full-CAN Transmit Objects and several Basic-CAN Transmit Objects as well as one Basic-CAN Transmit Object and several Full-CAN Transmit objects etc.

▽

△

Dependencies:	[SRS_Can_01058], [SRS_Can_01049]
Supporting Material:	–

⌋

6.2.1.2 Initialization

[SRS_Can_01041] The CAN Driver shall implement an interface for initialization

Upstream requirements: RS_BRF_01704, RS_BRF_01136

⌈

Description:	The CAN Driver shall implement an interface for initialization. This service shall initialize all module global variables and all Registers of the CAN Hardware Unit and its Controller(s). This function shall only be called once during startup
Rationale:	Basic functionality.
Use Case:	A CAN Hardware Unit has registers that must be set according the static configuration. Some register values belong to one single CAN controller some influence the complete unit
Dependencies:	–
Supporting Material:	

⌋

[SRS_Can_01042] The CAN Driver shall support dynamic selection of configuration sets

Upstream requirements: RS_BRF_01704, RS_BRF_01152

⌈

Description:	The CAN Driver shall support the dynamic selection of one static configuration set out of a list of configuration sets. This shall be done by a parameter passed via the initialization interface. Refer to CAN Driver SWS[1] for a detailed view of parameters. To switch to another configuration set shall only be possible if the CAN driver's state machine is in STOPPED mode. Hints: The selection of the appropriate configuration set itself as well as the way to incorporate the configuration sets into the ECU (Post-Build, Pre-Compile) are not affected by this requirement
Rationale:	Support of different configurations during runtime

▽

△

Use Case:	Use different configuration sets with e.g. different CAN IDs depending on different mounting positions of the ECU
Dependencies:	–
Supporting Material:	

⌋

6.2.1.3 Normal Operation

[SRS_Can_01043] The CAN Driver shall provide a service to enable/disable interrupts of the CAN Controller.

Upstream requirements: RS_BRF_01704

⌈

Description:	The CAN Driver shall offer services for enabling and disabling all interrupts generated by a CAN controller • Disabling means: Disable all interrupts of the related CAN Controller • Enabling means: Re-enable all interrupts which were disabled before
Rationale:	Basic functionality, ensure data consistency
Use Case:	Used to disable asynchronous interruptions by a CAN Driver event.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01059] The CAN Driver shall guarantee data consistency of received L-PDUs

Upstream requirements: RS_BRF_01704, RS_BRF_01632

⌈

Description:	The CAN Driver shall guarantee that the data inside a Hardware Object is not overwritten while it is copied
Rationale:	Basic functionality
Use Case:	A newly arrived message may overwrite the CAN Hardware buffer during the data is read out of the CAN Controller. This may lead to inconsistent data. Therefore the Driver shall ensure that inconsistent data is not copied.
Dependencies:	–

▽

△

Supporting Material:	–

⌋

[SRS_Can_01045] The CAN Driver shall offer a reception indication service.

Upstream requirements: RS_BRF_01704, RS_BRF_01064, RS_BRF_01544

⌈

Description:	The CAN Driver shall notify the CAN Interface about a successful reception. The notification is done by call of a static callback function implemented inside the CAN Interface. The Notification includes the following information: • CAN Identifier • DLC • CAN Hardware Object • Pointer to SDU data
Rationale:	Basic functionality, CAN Standards coverage
Use Case:	According the CAN Service primitive, the reception of a received CAN frame shall be indicated to the next upper layer. This Service here is used by the CAN Interface (on indication it notifies the L-SDU Router to route the notification to next upper layer and the destination upper layer copies the received data)
Dependencies:	[SRS_Can_01003]
Supporting Material:	

⌋

[SRS_Can_01049] The CAN Driver shall provide a dynamic transmission request service

Upstream requirements: RS_BRF_01704, RS_BRF_01544

⌈

Description:	The CAN Driver API shall provide a dynamic transmission request service (called by CAN Interface). The DLC and ID of the L-PDU are given as parameter. The CAN Interface provides following parameters: • CAN Hardware Object Handle (implies the CAN Controller) • L-PDU: – Pointer L-SDU source – CAN Identifier – DLC

▽

△

Rationale:	Basic functionality, CAN Standards coverage
Use Case:	Basic-CAN transmit hardware objects
Dependencies:	[SRS_Can_01008]
Supporting Material:	–

⌋

[SRS_Can_01051] The CAN Driver shall provide a transmission confirmation service

Upstream requirements: RS_BRF_01704, RS_BRF_01544

⌈

Description:	The CAN driver shall notify the CAN Interface about a successful transmission. Successful transmission means in this case, that at least one receiver acknowledged the CAN frame and it has not been disturbed by an error. The notification is done by call of a static call-back function implemented inside the CAN Interface
Rationale:	Basic functionality, CAN Standards coverage
Use Case:	According the CAN Service primitive, the transmission of a CAN frame shall be confirmed.
Dependencies:	[SRS_Can_01009]
Supporting Material:	ISO11898[7] Section 6.3.3 'Recovery management

⌋

[SRS_Can_01053] The CAN Driver shall provide a service to change the CAN controller mode.

Upstream requirements: RS_BRF_01704

⌈

Description:	The CAN Driver shall provide a service to change the mode of the specified CAN controller. The following states shall be supported: • UNINIT - The CAN controller is not configured, typically the registers are in reset state • STOPPED - The CAN controller is configured but does not take part in the CAN communication • STARTED - The CAN controller is up and running • SLEEP - The CAN controller is in sleep mode. The corresponding CAN Driver SWS describes the possible state transitions in detail. All necessary HW-initializations for the respective mode transition are done inside thisservice.
Rationale:	Basic functionality
Use Case:	The CAN controller may be initialized for low power consumption in sleep mode. This is done with this service for SLEEP transition. In case of bus-off, the controller may be set in UNINIT state (typically reset of controller) and set to running later on.
Dependencies:	[SRS_Can_01027]
Supporting Material:	–

⌋

[SRS_Can_01054] The CAN Driver shall provide a notification for controller wake-up events

Upstream requirements: RS_BRF_01704, RS_BRF_01664

⌈

Description:	The CAN driver module shall notify the Service Layer in case of a wake-up interrupt of the CAN controller. The notification is done by a call of a static callback function which is specified by ECU StateManager, but implemented by Complex Driver or so called "Integration Code". This functionality shall only be implemented, if CAN Hardware unit supports sleep mode and a specific wakeup interrupt is available. Even if the CAN Hardware supports it, this feature shall be Pre-Compile-Time configurable.
Rationale:	Basic functionality

▽

△

Use Case:	Any wakeup source is notified to the ECU StateManager. The ECU StateManager forwards this notification to the responsible module (typically the CAN Interface), which checks the wakeup source.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01122] The CAN driver shall support the situation where a wakeup by bus occurs during the same time the transition to standby/sleep is in progress

Upstream requirements: RS_BRF_01704, RS_BRF_01664

⌈

Description:	Wakeup by bus is always asynchronous to the internal transition to sleep. In worst case, the wakeup occurs during the transition to sleep. This situation must be covered by the software design and explicitly tested for each ECU. Assuming this worst case, the driver shall raise the Wake-up Notification immediately after the API to enter the standby/sleep mode has finished. Hint: In case the ECU hardware has the capability to notify one wakeup reason from different hardware components e.g. Transceiver and Controller, it's up to the system configuration to select one source
Rationale:	Safe wakeup and sleep handling.
Use Case:	All busses with a wakeup by bus are affected.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01132] The CAN driver shall be able to detect notification events message object specific by CAN-Interrupt and polling

Upstream requirements: RS_BRF_01704

⌈

Description:	Dependent on configuration the detection of any reception, transmission or error event shall be done by release a CAN Interrupt and by Polling through the CAN driver. Both mechanisms shall be configurable for each message object if supported by CAN Hardware
Rationale:	Polling the CAN HW globally leads to the problem, that the polling rate belongs to the CAN message with the shortest cycle time, which may result in very high runtimes. Notification by interrupt offers the possibility to react real time. This is useful especially on messages with very short cycle times.

▽

△

Use Case:	Gateway / CCP / Network Layer <=> Intersystem communication. Time triggered Complex Drivers, which have strong restrictions to guarantee fixed reaction times and which shall ensure predictable behavior.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01134] The CAN Driver shall support multiplexed transmission

Upstream requirements: RS_BRF_01704

⌈

Description:	The CAN Driver shall support multiplexed transmission if supported by the underlying CAN Controller Definition of 'multiplexed transmission': Three TX HW objects are represented as one Transmit entity (Hardware Object Handle) to the upper layer. This avoids gaps between consecutive sending of L-PDUs. This feature option shall only be implemented when the CAN Hardware fulfills the following requirements: [The three HW objects are represented as single register set OR the hardware provides registers that identify a free buffer] AND [The L-PDUs are sent out in the order of their priority]
Rationale:	Outer priority inversion can be avoided
Use Case:	Basic-CAN transmit hardware objects
Dependencies:	[SRS_Can_01058]
Supporting Material:	–

⌋

[SRS_Can_01147] The CAN Driver shall not support remote frames

Upstream requirements: RS_BRF_01704

⌈

Description:	The CAN driver shall not transmit messages triggered by remote transmission requests. The CAN driver shall initialize the CAN HW to ignore any remote transmission requests.
Rationale:	Remote transmission requests are not used in automotive area.
Use Case:	See rational
Dependencies:	–

▽

△

Supporting Material:	–

⌋

[SRS_Can_01161] The CAN Driver shall not support remote frames

Upstream requirements: RS_BRF_01712

⌈

Description:	The CAN driver shall be able to operate with both classic CAN and CAN FD frames on one CAN Controller if supported by CAN Hardware.
Rationale:	CAN (FD) standard coverage
Use Case:	CAN FD frames support up to 64 bytes per frame at a higher baud rate which might be required by some projects.
Dependencies:	–
Supporting Material:	ISO 11898-1[7]

⌋

[SRS_Can_02001] The CAN Driver shall support CAN XL

Upstream requirements: RS_BRF_01712

⌈

Description:	The CAN driver shall support CAN XL besides CAN 2.0 and CAN FD.
Rationale:	CAN XL provides higher bandwidth than CAN 2.0 or CAN FD and allows native tunneling of Ethernet frames.
Use Case:	Transmission and reception of CAN XL frames of all SDU types, handling of CAN and Ethernet bus states.
Dependencies:	Requires an extended CAN transceiver driver that supports the CAN XL baud rates and Ethernet link state handling.
Supporting Material:	CiA 611 (see [1])

⌋

[SRS_Can_01167] The CAN Driver shall provide a function to return the current CAN controller error state ⌈

Description:	The function shall return the current driver state ACTIVE, PASSIVE and BUSOFF.
Rationale:	Setting DTC when entering CAN passive or bus-off state.

▽

△

Use Case:	User of the CAN driver require setting a DTC in case of CAN passive or bus-off state.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01170] The CAN Driver shall provide a function to return the current CAN controller Rx and Tx error counters ⌈

Description:	The CAN driver shall report the current Rx and Tx error counters via dedicated functions.
Rationale:	The error counters are available in most CAN controllers, and AUTOSAR should provide a standardized access to this information.
Use Case:	Provide information about current state of a CAN bus for diagnostic purposes.
Dependencies:	–
Supporting Material:	Concept 634 "Bus Mirroring"

⌋

[SRS_Can_02006] CAN Driver Service for Fetching L-PDU Message

Status: DRAFT
Upstream requirements: RS_BRF_01704

⌈

Description:	The Can Driver shall provide an service to fetch the message state of an L-PDU.
Rationale:	L-PDUs encapsulated with [8, IEEE1722] protocol require the message state which is added to the ACF header.
Use Case:	Support for IEEE1722 tunneling of CAN frames as Time Synchronous and Non-Time Synchronous Control Frames Format
Dependencies:	–
Supporting Material:	[8, IEEE1722]

⌋

6.2.1.4 Shutdown Operation

[SRS_Can_01166] The CAN Driver shall implement an interface for de-initialization ⌈

Description:	The CAN Driver shall implement an interface for de-initialization. This service shall de-initialize the CAN Hardware Unit and its Controller(s).
Rationale:	Basic Functionality
Use Case:	A CAN Hardware Unit shall be re-configured with a new configuration set without the need for an ECU reset.
Dependencies:	–
Supporting Material:	–

⌋

6.2.1.5 Fault Operation

[SRS_Can_01055] CAN Driver shall provide a notification for bus-off state

Upstream requirements: RS_BRF_01704, RS_BRF_01408, RS_BRF_01664

⌈

Description:	The CAN driver shall notify the CAN Interface if the CAN Controller goes in bus-off state. The notification is done by call of a static callback function implemented inside the CAN Interface.
Rationale:	Basic Functionality
Use Case:	Any state transition is notified to the CAN Interface. The CAN Interface forwards this notification to the responsible layer.
Dependencies:	[SRS_Can_01029]
Supporting Material:	–

⌋

[SRS_Can_01060] The CAN driver shall not recover from bus-off automatically

Upstream requirements: RS_BRF_01704, RS_BRF_01664

⌈

Description:	The bus-off recovery shall be software driven. If an automatic bus-off recovery is implemented in the hardware it has to be suppressed by software e.g. force CAN controller to reset state within the bus off interrupt service routine

▽

△

Rationale:	Basic Functionality
Use Case:	A software-controlled recovery allows other nodes to communicate without the damaged node disturbing the bus for some time period
Dependencies:	–
Supporting Material:	–

⌋

6.2.2 CAN Interface (Hardware Abstraction)

The CAN Interface provides standardized interfaces to provide the communication with the CAN bus system of an ECU. The APIs are independent from the specific CAN Controllers and Transceivers and their access through the responsible Driver layer. The CAN Interface is able to access one or more CAN Drivers and CAN Transceiver Drivers via one uniform interface.

For a detailed functional description and interface definition see CAN Interface Specification [[2]].

6.2.2.1 Configuration

[SRS_Can_01015] The CAN Interface configuration shall be able to import information from CAN communication matrix.

Upstream requirements: RS_BRF_01704

⌈

Description:	The static configuration of the CAN Interface shall be based on information from the CAN communication matrix. The following information shall be extracted from the CAN communication matrix: • Individual RX L-PDUs for each CAN Controller - identified by CAN ID • RX L-PDU ranges for each CAN Controller • All TX L-PDUs for each CAN Controller - identified by CAN ID • TX L-PDU ranges for each CAN Controller • Upper layer client for each L-PDU (-range) • DLC for each L-PDU (-range) The configuration parameters shall be allowed to be of types Pre-Compile, Link-Time or Post-Build

▽

△

Rationale:	Common Database for CAN Network
Use Case:	The communication matrix is used to describe all messages in a network and their sender and receiver. This information can be taken to configure the software filter algorithm, the DLC check and the notifications for the CAN Interface.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01016] The CAN Interface shall have an interface to the static configuration information of the CAN Driver

Upstream requirements: RS_BRF_01704

⌈

Description:	The CAN Interface and its code configurator/generator shall be able to read the CAN Driver configuration inside the ECU configuration description
Rationale:	Flexibility and scalability
Use Case:	Optimization of software filtering according configured hardware filters
Dependencies:	[SRS_Can_01036], [SRS_Can_01039]
Supporting Material:	–

⌋

[SRS_Can_01018] The CAN Interface shall have an interface to the static configuration information of the CAN Driver

Upstream requirements: RS_BRF_01704

⌈

Description:	All L-PDUs that are not filtered by HW-Filters and are not defined as receive L-PDUs in the network database need to be rejected by a filter implemented in software.
Rationale:	Basic functionality
Use Case:	Messages that shall not be received by the ECU, but could not be filtered by hardware filters, shall be filtered by software in the CAN Interface.
Dependencies:	[SRS_Can_01037], [SRS_Can_01004], [SRS_Can_01039]
Supporting Material:	–

⌋

[SRS_Can_01019] It shall be Pre-Compile-Time configurable whether a DLC check is performed or not

Upstream requirements: RS_BRF_01704

⌈

Description:	It shall be Pre-Compile-Time configurable whether the DLC check global for each CAN controller is performed
Rationale:	Basic functionality
Use Case:	Turning off the DLC check improves the exchangeability of older ECUs, where IDs stay the same but SDU length differs
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01020] The TX-Buffer shall be statically configurable

Upstream requirements: RS_BRF_01704

⌈

Description:	It shall be configurable Pre-Compile-Time, whether one or no buffer per L-PDU shall be available
Rationale:	–
Use Case:	Different properties are necessary to realize different variants of ECUs
Dependencies:	[SRS_Can_01011]
Supporting Material:	–

⌋

6.2.2.2 Initialization

[SRS_Can_01021] CAN The CAN Interface shall implement an interface for initialization

Upstream requirements: RS_BRF_01704, RS_BRF_01136

⌈

Description:	The CAN Interface shall implement an interface for initialization. This service shall initialize all module global variables.
Rationale:	Basic functionality.

▽

△

Use Case:	A CAN Interface has static variables that need to be initialized, before the CAN Interface can be used.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01022] The CAN Interface shall support the selection of configuration sets

Upstream requirements: RS_BRF_01704, RS_BRF_01136

⌈

Description:	The CAN Interface shall support the selection of one configuration set out of a list of different static configuration sets. This shall be done by a parameter passed via the initialization interface. This is typically done once during startup
Rationale:	Support of different configurations during runtime
Use Case:	Another module (independently from CanIf) checks the startup conditions e.g. depending on the mounting position in the car, selects the appropriate configuration set. This is then passed to the CanIf.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01023] The CAN Interface shall be initialized in a defined way.

Upstream requirements: RS_BRF_01704, RS_BRF_01136

⌈

Description:	The CAN Interface shall be initialized in the following sequence: 1. Initialize global variables 2. Reset flags This sequence has to be executed in this order, because the CAN Interface has to be operable before CAN Driver (and thus the communication started)
Rationale:	Defined initialization sequence without side effects.
Use Case:	Power on reset
Dependencies:	–
Supporting Material:	–

⌋

6.2.2.3 Normal Operation

[SRS_Can_01002] The CAN Interface shall be responsible for the dispatching of the received PDUs

Status: OBSOLETE
Use instead: SRS_Can_02004
Upstream requirements: RS_BRF_01704

⌈

Description:	The CAN Interface knows which upper layer is the addressee of a successfully received L-PDU and makes a decision to which layer it belongs. That's why the CAN Interface can redirect sequential L-PDU to its destination
Rationale:	Basic functionality
Use Case:	Provide access to received CAN data by different upper layers
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_02004] CanIf Forwarding of L-PDUs to L-SDU Router

Status: DRAFT
Upstream requirements: RS_BRF_01704

⌈

Description:	The CanIf shall forward L-PDUs to the L-SDU Router. Additional Information: The CanIf must have exactly one upper layer module which perform the PDU routing with frame specific information. Therefore a L-PDU must be split in frame specific information and payload. Frame specific information must be added to meta data and payload shall be assigned to the according PDU ID
Rationale:	Basic functionality
Use Case:	• Provide access to received CAN data by different upper layers via the L-SDU Router • Support for IEEE1722 tunneling of CAN frames as Time Synchronous and Non-Time Synchronous Control Frames Format
Dependencies:	–
Supporting Material:	[8, IEEE1722]

⌋

[SRS_Can_01003] The appropriate higher communication stack shall be notified by the CAN Interface about an occurred reception

Status: OBSOLETE
Use instead: SRS_Can_02005
Upstream requirements: RS_BRF_01704, RS_BRF_01544

⌈

Description:	The CAN driver will indicate each successfully received L-PDU. The appropriate higher communication stack shall be notified by the CAN Interface about an occurred reception. This routing of an indication event is the task of the CAN Interface. An indication is only a notification, where no data is transferred. The information which L-PDU has been received shall be part of the indication
Rationale:	Basic functionality, CAN Standards Coverage
Use Case:	According the CAN Service primitive, the reception of a received CAN frame shall be indicated to the next upper layer.
Dependencies:	[SRS_Can_01045]
Supporting Material:	–

⌋

[SRS_Can_02005] The appropriate higher communication stack shall be notified by the CAN Interface via the L-SDU Router about an occurred reception

Status: DRAFT
Upstream requirements: RS_BRF_01704, RS_BRF_01544

⌈

Description:	The CAN driver will indicate each successfully received L-PDU. The appropriate higher communication stack shall be notified by the CAN Interface via the L-SDU Router about an occurred reception. This routing of an indication event is the task of the L-SDU Router. An indication is only a notification, where no data is transferred. The information which L-PDU has been received shall be part of the indication
Rationale:	Basic functionality, CAN Standards Coverage
Use Case:	According the CAN Service primitive, the reception of a received CAN frame shall be indicated to the L-SDU Router.
Dependencies:	[SRS_Can_01045]
Supporting Material:	–

⌋

[SRS_Can_01114] Data Consistency of L-PDUs to transmit shall be guaranteed

Upstream requirements: RS_BRF_01704, RS_BRF_01632

⌈

Description:	During copying of transmit data it must be prevented that the corresponding memory area is overwritten by upper layer
Rationale:	Data Consistency
Use Case:	Upper Layer writes to a data area that is at the same read out for a CAN transmission. This will lead to inconsistent data and therefore has to be prevented
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01004] Software filtering shall be implemented by the CAN Interface

Upstream requirements: RS_BRF_01704, RS_BRF_01608

⌈

Description:	A L-PDU filtering based on the CAN Identifier shall be implemented by the CAN Interface. In case the received L-PDU did not pass the software filter, it will not further be processed. The upper layer will not be notified
Rationale:	Basic functionality
Use Case:	Messages that shall not be received by the ECU, but could not be filtered by hardware filters, shall be filtered by software in the CAN Interface.
Dependencies:	[SRS_Can_01015], [SRS_Can_01018], [SRS_Can_01037], [SRS_Can_01039]
Supporting Material:	–

⌋

[SRS_Can_01005] The CAN Interface shall perform a check for correct DLC of received PDUs

Upstream requirements: RS_BRF_01704

⌈

Description:	The CAN Interface shall check the DLC of received L-PDUs that have passed the SW filter. The DLC shall be larger or equal to the configured L-PDU length. In case the received L-PDU did not pass the DLC check, it shall not be further processed
Rationale:	Basic functionality
Use Case:	Avoid data inconsistency because of incomplete L-SDU

▽

△

Dependencies:	[SRS_Can_01015]
Supporting Material:	–

⌋

[SRS_Can_01006] The CAN Interface shall provide a service to enable/disable L-PDU reception per CAN Controller

Upstream requirements: RS_BRF_01704, RS_BRF_01680

⌈

Description:	The API of the CAN Interface shall provide a service to enable/disable the reception of all incoming L-PDUs belonging to one CAN Controller, that normally would cause a receive indication (and data copy). In case the received L-PDU is disabled, it will not further be processed. The upper layer will not be notified. This service is directly tunneled to the appropriate CAN driver
Rationale:	Basic functionality
Use Case:	The COM Manager must be capable to suppress all reception event of the corresponding CAN network It is the complementary functionality to switching on/off the transmission path.
Dependencies:	[SRS_Can_01013]
Supporting Material:	–

⌋

[SRS_Can_01007] The CAN Interface shall dispatch the transmission request by an upper layer module to the desired CAN controller

Upstream requirements: RS_BRF_01704, RS_BRF_01544

⌈

Description:	In case the CAN Hardware Unit consists of more than one CAN controller the CAN Interface shall dispatch the transmission request by an upper layer module to the desired CAN controller
Rationale:	Basic functionality
Use Case:	More than one on-chip CAN Controller on one ECU.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01008] The CAN Interface shall provide a transmission request service

Upstream requirements: RS_BRF_01704, RS_BRF_01544

⌈

Description:	The CAN Interface API shall provide a transmission request service. The L-PDU is either forwarded to the CAN Driver or stored in the TX Buffer
Rationale:	Basic functionality, CAN Standards Coverage
Use Case:	According the CAN Service primitive, a service for transmission shall be provided.
Dependencies:	[SRS_Can_01011], [SRS_Can_01020]
Supporting Material:	–

⌋

[SRS_Can_01009] The CAN Interface shall provide a transmission confirmation dispatcher

Upstream requirements: RS_BRF_01704, RS_BRF_01544

⌈

Description:	The CAN Interface has to notify the appropriate upper layer modules about successful transmission. Therefore the CAN Interface has to dispatch the transmit confirmation after confirmation of the CAN driver. It shall be statically configurable per PDU if the confirmation shall be forwarded to upper layer or not
Rationale:	Basic functionality, CAN Standards Coverage
Use Case:	According the CAN Service primitive, the transmission of a CAN frame shall be confirmed.
Dependencies:	[SRS_Can_01051]
Supporting Material:	–

⌋

[SRS_Can_01011] The CAN Interface shall provide a transmit buffer

Upstream requirements: RS_BRF_01704, RS_BRF_01544

⌈

Description:	The CAN Interface shall buffer pending transmit requests only: • if the CAN driver rejected the preceded transmit request because of not available hardware resources • in case that a pending transmit request was cancelled in the CAN Driver The transmit buffer shall provide the following functionality: • each transmit L-PDU shall have exactly one reference to a buffer container • the size of buffer container defines the number of L-PDU's which can be buffered • if the buffer size is 0 it means no CanIf buffering will be made • each Buffer container shall have 1...n references to logical hardware transmit objects(HTH's) (which will be used for transmission) • one HTH has exactly one reference to a buffer • the buffer shall be flushed only in case of reaching the "Tx Offline" state • the buffer shall have a priority order and shall not store more than one instance of a L-PDU • in case of buffer overflow the transmission service shall return "Not OK" • During Tx confirmation the L-PDU with the highest priority shall be forwarded to the CAN driver. The priority is defined by the CAN Identifier that belongs to the transmit L-PDU. Only the newest instance of an L-PDU shall be stored in an own buffer and older ones shall be overwritten • There shall be a configuration option to define the buffer fixed to 8 Bytes. It shall be Pre-Compile-Time configurable whether the CanIf provides transmit buffers or not.
Rationale:	Basic functionality, limited resources for Tx-buffering
Use Case:	A message might not be sent out immediately because messages with higher priority are pending. Buffering of one instance per PDU is needed to ensure minimal delay times per L-PDU.
Dependencies:	[SRS_Can_01020],[SRS_Can_01008]
Supporting Material:	–

⌋

[SRS_Can_01013] The CAN Interface shall provide a Tx-L-PDU enable/disable service per CAN Controller

Upstream requirements: RS_BRF_01704, RS_BRF_01680

⌈

Description:	NMs require an additional software service to lock and unlock the transmission of outgoing L-PDUs belonging to one CAN Controller. This functionality has to be placed in the CAN Interface. Decision by WP Architecture.
Rationale:	Basic functionality
Use Case:	–
Dependencies:	[SRS_Can_01006]
Supporting Material:	–

⌋

[SRS_Can_01027] The CAN Interface shall provide a service to change the CAN Controller mode.

Upstream requirements: RS_BRF_01704, RS_BRF_01664

⌈

Description:	The CAN Interface shall provide a service to change the mode of the specified CAN controller. This service is typically called by the NM with respect on view of a physical channel. Restriction: a physical channel is only represented by one CAN controller. The following modes shall be supported: • UNINIT • STARTED • STOPPED • BUSOFF (not reachable by software) • SLEEP All necessary initializations for the respective mode transition is done inside the CAN Driver. Possible state transitions are described in the corresponding CAN Driver SWS
Rationale:	Basic functionality
Use Case:	This service represents the interface for the CAN Driver Mode Select service.
Dependencies:	[SRS_Can_01053]
Supporting Material:	–

⌋

[SRS_Can_01028] The CAN Interface shall provide a service to query the CAN controller state

Upstream requirements: RS_BRF_01704, RS_BRF_01664

⌈

Description:	The CAN Interface shall provide a service to query the CAN controller state. Please refer to the CAN Interface SWS document for details of the possible states. Hint: With this service the internal state of CAN Interface is polled. The actual hardware state may differ in some situations for a certain time
Rationale:	Basic functionality
Use Case:	May be used if CAN Controller doesn't provide interrupt service.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01151] The CAN Interface shall provide a service to check for a CAN Wake-up event.

Upstream requirements: RS_BRF_01704, RS_BRF_01104, RS_BRF_01680

⌈

Description:	The CAN Interface module shall provide a service to check for a CAN wake-up source in case of a CAN wake-up event. This service queries the CAN controllers and CAN transceivers by the Driver modules in order to find the wake-up source.
Rationale:	Basic functionality
Use Case:	A wake up by CAN can be recognized by an ECU in different ways: polling, CAN Controller interrupt, CAN Transceiver interrupt. In each case the ECU StateManager will need this service to check the CAN interface for the Wake-up Source that caused the Wake-up. For further details on the use case see figures 33-35 in document ECU StateManager.
Dependencies:	[SRS_Can_01032]
Supporting Material:	–

⌋

[SRS_Can_01032] The CAN Interface shall report a wake-up notification to the ECU StateManager

Upstream requirements: RS_BRF_01704, RS_BRF_01664, RS_BRF_01680

⌈

Description:	After the CAN Interface module checks the CAN controller and the CAN transceiver for wake-up events, it should notify the ECU StateManager about the event and source that caused the wake-up.
Rationale:	Basic functionality
Use Case:	A wake up by CAN can be recognized by an ECU in different ways. In each case the ECU StateManager will need this notification in order to activate the correct CAN Controller for Wake-up validation. For further details on the use case see figures 33-35 in document ECU StateManager.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01061] The CAN Interface shall provide dynamic TX Handles

Upstream requirements: RS_BRF_01704

⌈

Description:	The CAN Interface shall provide dynamic TX Handles which can be allocated by the upper layers. It shall be possible to change the ID and DLC of a Dynamic TX Handle by the upper layers. It shall be Pre-Compile-Time configured whether to use this feature or not
Rationale:	Communication with a blank or invalidL-PDU ID table or direct upper layer control of the CAN identifier.
Use Case:	Dynamically calculated TX IDs. Only ranges of IDs are allowed that are known in the network. Typically used by TP, where the target address is coded within the CAN Identifier. The target address can't be statically defined
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01159] The CAN Interface shall provide dynamic RX Handles

Upstream requirements: RS_BRF_01704, RS_BRF_01728, RS_BRF_01736

⌈

Description:	The CAN Interface shall provide dynamic RX handles which can beallocated by the upper layers. The ID and DLC of a dynamic RX handle will be provided to the upper layers. It shall be Pre-Compile-Time configured whether to use this feature or not.
Rationale:	Access to the CAN identifier by upper layers.
Use Case:	Dynamically evaluated RX IDs. Typically used by TP or J1939, where the target and/or source addresses are coded within the CAN Identifier.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01130] Receive Status Interface of CAN Interface

Upstream requirements: RS_BRF_01704

⌈

Description:	The CAN Interface shall additionally provide an Interface that the notification state of messages can be polled by upper layers
Rationale:	Flexible integration Avoid strong coupling and dependencies Deterministic behavior of upper layers for time triggered behavior
Use Case:	The completion of a CAN transmit request command can be signaled not only by a callback function, now also by a status information, which is accessible via the module interface. A fault occurred during the CAN transmit request (bus is blocked, CAN controller is defective) can be signalized via an error hook.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01131] The CAN Interface module shall provide the possibility to have polling and callback notification mechanism in parallel

Upstream requirements: RS_BRF_01704, RS_BRF_01544

⌈

Description:	Beside callback notification mechanisms at the same time a "Read Message Data" and "Read Message Status" API shall be able to be used.
Rationale:	It shall be possible, that upper layers can adapt the access to new data and status of received CAN messages according to their needs and they are not dependent to the network traffic. Different CAN Interface clients have different needs for latencies (notification mechanism provide a small latency time, a polling mechanism provides a big latency time). Thus it shall be possible, to differentiate the read data and notification mechanisms between the different CAN message to be received.
Use Case:	Gateway / CCP / Network Layer <=> Intersystem communication. Time triggered Complex Drivers, which have strong restrictions to guarantee fixed reaction times and which shall ensure predictable behavior.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01136] The CAN Interface module shall provide a service to check for validation of a CAN wake-up event

Upstream requirements: RS_BRF_01704, RS_BRF_01680, RS_BRF_01664

⌈

Description:	The CAN Interface module shall provide a service to check for validation of a CAN wake-up event (see [SRS_Can_01032]). It notifies the ECU StateManager about a validated wake-up event, only if a message was received correctly on the CAN bus where the wake-up event was detected.
Rationale:	Reduce power consumption
Use Case:	The Wake-up validation service should be called by the ECU Statemanager after the corresponding CAN Tranceiver was set to normal mode and the CAN Controller was started. During validation incoming messages must not be forwarded by the CAN Interface to upper layers, since the corresponding L-PDU channel groups should still be disabled (offline).
Dependencies:	[SRS_Can_01032]
Supporting Material:	–

⌋

[SRS_Can_01129] The CAN Interface module shall provide a procedural interface to read out data of single CAN messages by upper layers (Polling mechanism)

Upstream requirements: RS_BRF_01704, RS_BRF_01544

⌈

Description:	After getting information about new received data (by call get status interface SRS_SPAL_00157) the upper layer must be able to read out data. Thus the CAN Interface shall provide a corresponding API ('ReadMessageData()') to read out data of received CAN messages. The described function shall be Pre-Compile-Time selectable
Rationale:	Flexibility (The layer above should have the possibility to decide when and if data should be transferred (data flow is controlled by upper layer)) Avoid strong coupling and dependencies (see Rationale of BSW 157) There are applications with deterministic behavior inside time triggered software systems. Deterministic behavior can only be ensured if these applications aren't interrupted by bus events
Use Case:	The notification of the completion of a CAN message reception event can be used to read out the data at point of time the upper layers needs it. Using the API the data are accessed either from the CAN Hardware buffer or from the shadow buffer of the CAN driver. This intermediate buffer needed e.g. data normalization for the 'GetMessageData()' API shall be configurable for each CAN Rx Identifier.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01140] The CAN Interface shall support both Standard (11bit) and Extended (29bit) Identifiers

Upstream requirements: RS_BRF_01704

⌈

Description:	The CAN Interface shall support Standard and Extended Identifiers. It shall be configurable per network whether Standard or Extended Identifiers are supported
Rationale:	Standard CAN 2.0b functionality
Use Case:	–
Dependencies:	[SRS_Can_01141]
Supporting Material:	–

⌋

[SRS_Can_01141] The CAN Interface shall support both Standard (11bit) and Extended (29bit) Identifiers at same time on one network

Upstream requirements: RS_BRF_01704

⌈

Description:	This requirement describes an implementation variant beyond [SRS_Can_01140]: The CAN Interface shall be able to support Standard and Extended Identifiers at same time on one network (=mixed mode support). Due to significant consequences on code efficiency and complexity, this feature shall be optional. In case of not purchasing this feature, [SRS_Can_01140] is still valid.
Rationale:	–
Use Case:	Usage of cheap Basic CAN Controllers in CAN networks with both Identifier types
Dependencies:	[SRS_Can_01036]
Supporting Material:	–

⌋

[SRS_Can_01153] The Tx-Filter shall ensure, that the first message which is sent on the bus is a Wakeup Frame (WUF) in the case of partial networking

Upstream requirements: RS_BRF_01704, RS_BRF_01680

⌈

Description:	If a L-PDU gets activated for transmission the Tx-Filter shall be switched into blocking mode. If a Tx-Filter is in blocking mode, then all L-PDUs shall be discarded, except the Wakeup Frame (WUF). If a L-PDU is in blocking mode and the Wakeup Frame (WUF) gets transmitted it shall be forwarded to the lower layer. If the CAN-Interface receives a transmit notification of the WUF, the Tx-Filter shall be switched into pass mode. If the Tx-Filter is in pass mode, then all L-PDUs shall be forwarded to the lower layer. The Tx-Filter shall not be activated during Bus-Off mode.
Rationale:	If partial networking is used the ECU must secure that the first message on the bus is the Wakeup Frame (WUF).
Use Case:	Starting communication from BusSleep Mode, PrepareBusSleep Mode, BusOff
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01181] If partial networking is used, the ECU shall secure that the first message on the bus is the wakeup frame. ⌈

Description:	If partial networking is used, the ECU shall secure that the first message on the bus is the wakeup frame. This requirement will be implemented in CanIf.
Rationale:	If all ECUs on the bus use partial networking, they use the CAN transceiver with the partial networking extensions. These transceivers only wake up after receiving the Wakeup Frame.
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01182] CanIf shall provide an optional channel-specific TX filter ⌈

Description:	If a L-PDU gets activated for transmission the Tx-Filter shall be switched into blocking mode. CanIf shall provide an optional channel-specific TX filter. In blocking mode, the filter shall only pass transmission of wakeup frames. In pass mode the filter shall pass every PDU transmitted by an upper layer.
Rationale:	–
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01183] CanIf shall provide the possibility to initiate clear and check wake-up flags in the transceiver ⌈

Description:	CanIf shall provide the possibility to initiate clear and check wake-up flags in the transceiver.
Rationale:	–
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01158] The CAN stack shall provide a TX offline active mode for ECU passive mode

Upstream requirements: RS_BRF_01704

⌈

Description:	The CAN stack shall provide a tx offline active mode to allow ECU Passive Mode.
Rationale:	ECU Passive Mode is used for disabling all Tx Requests by "simulating" successfull transmit requests towards applications.
Use Case:	Diagnostics, switching all transmissions off temporarily
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01160] Padding of bytes due to discrete CAN FD DLC

Upstream requirements: RS_BRF_01712

⌈

Description:	Unused bytes caused by discrete DLC for CAN FD frames > 8 bytes shall be padded.
Rationale:	CAN FD frames support up to 64 bytes by using only 4 bit DLC to indicate payload length. However, the length for frames > 8 bytes can be configured to 12, 16, 20, 24, 32, 48, and 64 bytes. If a PDU does not exactly match these configurable sizes the unused bytes shall be padded.
Use Case:	PDUs are declared to different sizes than the discrete DLC for CAN FD. Sizes up to next discrete DLC must be padded to avoid misinterpretation while reception.
Dependencies:	–
Supporting Material:	ISO 11898-1[7]

⌋

[SRS_Can_01162] CAN Interface shall support classic CAN and CAN FD frames

Upstream requirements: RS_BRF_01712

⌈

Description:	The CAN Interface shall support classic CAN and CAN FD L-PDUs. It shall be configurable per L-PDU whether classic CAN or CAN FD frames are assigned.
Rationale:	CAN (FD) standard functionality

▽

△

Use Case:	CanIf has to differ between CAN and CAN FD L-PDUs to allow adequate processing in upper layers e.g. CanTp.
Dependencies:	[SRS_Can_01061]
Supporting Material:	ISO 11898-1[7]

⌋

[SRS_Can_02003] The CAN Interface shall support CAN XL frames

Upstream requirements: RS_BRF_01712

⌈

Description:	The CAN interface shall allow access to CAN XL drivers
Rationale:	CAN XL provides higher bandwidth than CAN 2.0 or CAN FD, and allows native tunneling of Ethernet frames.
Use Case:	Transmission and reception of all kinds of CAN XL frames apart from SDU Type 5 (mapped Ethernet) and bus state handling.
Dependencies:	Requires an extended CAN driver and CAN bus transceiver that support CAN XL.
Supporting Material:	CiA 611 (see [1])

⌋

[SRS_Can_01169] The CAN interface shall provide a function to return the current CAN controller error state ⌈

Description:	The function shall return the current driver state ACTIVE, PASSIVE and BUSOFF.
Rationale:	Setting DTC when entering CAN passive or bus-off state.
Use Case:	User of the CAN driver require setting a DTC in case of CAN passive or bus-off state.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01171] The CAN Interface shall provide a function to return the current CAN controller Rx and Tx error counters ⌈

Description:	The CAN interface shall report the current Rx and Tx error counters via dedicated functions.
Rationale:	The error counters are available in most CAN controllers, and AUTOSAR should provide a standardized access to this information.
Use Case:	Provide information about current state of a CAN bus for diagnostic purposes.
Dependencies:	–
Supporting Material:	Concept 634 "Bus Mirroring"

⌋

[SRS_Can_01172] The CAN Interface shall provide a function to provide received and transmitted frames to the Bus Mirroring ⌈

Description:	If enabled, the CAN interface shall report all frames received and transmitted by one CAN controller to the Bus Mirroring.
Rationale:	This functionality should reside in the CAN interface, because the CAN interface abstracts from the different CAN driver modules, and still has access to all CAN frames that the CAN driver handles.
Use Case:	Mirroring of CAN bus traffic for diagnostic purposes.
Dependencies:	–
Supporting Material:	Concept 634 "Bus Mirroring"

⌋

6.2.2.4 Shutdown Operation

[SRS_Can_01168] The CAN Interface shall implement an interface for de-initialization ⌈

Description:	The CAN Interface shall implement an interface for de-initialization. This service shall put the module in a state that accepts a subsequent initialization call.
Rationale:	Basic functionality
Use Case:	A CAN Stack shall be re-configured with a new configuration set without the need for an ECU reset.
Dependencies:	–
Supporting Material:	–

⌋

6.2.2.5 Fault Operation

[SRS_Can_01029] The CAN Interface shall report bus-off state of a device to an upper layer

Upstream requirements: RS_BRF_01704, RS_BRF_01664

⌈

Description:	When the CAN Interface detects a bus-off state (by CAN Driver state change notification) a notification call-back function shall be called that is implemented in CAN State Manager.
Rationale:	Basic functionality
Use Case:	Any state transition is notified by the CAN Interface. The bus-off notification is typically handled by the CAN State Manager.
Dependencies:	[SRS_Can_01055]
Supporting Material:	–

⌋

6.2.3 CAN State Manager

6.2.3.1 Configuration

[SRS_Can_01143] The CAN State Manager shall support a configurable BusOff recovery time

Upstream requirements: RS_BRF_01704, RS_BRF_01664, RS_BRF_01600

⌈

Description:	The CAN State Manager shall control the BusOff recovery algorithm. The time between the CAN Controller detects a BusOff event and the restart of the communication shall configurable.
Rationale:	Basic functionality
Use Case:	Delay of communication after BusOff detection to overcome temporay bus disturbance.
Dependencies:	–
Supporting Material:	–

⌋

6.2.3.2 Initialization

[SRS_Can_01144] The CAN State Manager shall implement an interface for initialization.

Upstream requirements: RS_BRF_01704, RS_BRF_01664, RS_BRF_01600

⌈

Description:	The CAN State Manager shall provide an interface to initialize the communication mode at power-on. The communication mode for initialisation shall be configurable. It shall be possible to start up with full communication mode, with silent communication mode or with no communication mode.
Rationale:	Basic functionality
Use Case:	Different kinds of communication behaviours of ECUs after power-on (listen only until application needs full communication capability or immediate full communication capability).
Dependencies:	–
Supporting Material:	–

⌋

6.2.3.3 Normal Operation

[SRS_Can_01145] The CAN State Manager shall control the assigned CAN Devices

Upstream requirements: RS_BRF_01704

⌈

Description:	The CAN State Manager shall start and stop the CAN Devices and shall prepare them for sleep.
Rationale:	Complexity of CAN Interface is reduced
Use Case:	Split of data and control flow
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01184] When full communication is requested, CanSm shall enable pass mode on the CanIf TX filter ⌈

Description:	When full communication is requested, CanSm shall enable pass mode on the CanIf TX filter
Rationale:	–
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01185] CanSm shall provide the possibility to initiate clear and check wake-up flags in the transceiver ⌈

Description:	CanSm shall provide the possibility to initiate clear and check wake-up flags in the transceiver
Rationale:	–
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01186] CanSm shall support a valid PN shutdown sequence ⌈

Description:	CanSm shall support a validPN shutdown sequence (CAN CC STOP -> CAN TRCV STANBY -> CAN CC SLEEP)
Rationale:	–
Use Case:	
Dependencies:	–
Supporting Material:	–

⌋

6.2.3.4 Shutdown Operation

[SRS_Can_01164] The CAN State Manager shall implement an interface for de-initialization. ⌈

Description:	The CAN State Manager shall implement an interface for de-initialization. This service shall put the module in a state that accepts a subsequent initialization call.
Rationale:	Basic functionality
Use Case:	A CAN Stack shall be re-configured with a new configuration set without the need for an ECU reset.
Dependencies:	–
Supporting Material:	–

⌋

6.2.3.5 Fault Operation

[SRS_Can_01146] The CAN State Manager shall contain a CAN BusOff recovery algorithm for each used CAN Controller

Upstream requirements: RS_BRF_01704, RS_BRF_01664, RS_BRF_01600

⌈

Description:	The CAN State Manager shall control the CAN BusOff recovery by a algorithm. It shall report the production error "CAN BusOff" to the Diagnostic Event Manager. It shall report a specific "CAN BusOff"-production error for each configured CAN network, if recovery is not possible within a configurable time.
Rationale:	Network controller specific error and bus state management
Use Case:	See Rationale
Dependencies:	–
Supporting Material:	–

⌋

6.2.4 Transport Layer CAN

This chapter describes the requirements for the CAN Transport Layer [[4]].

The AUTOSAR CAN Transport Layer generally bases on the ISO 15765-2[9] and ISO 15765-4[10] specifications.

6.2.4.1 Configuration

[SRS_Can_01066] The AUTOSAR CAN Transport Layer shall be statically configurable to support either single or multiple connections in an optimizing way

Upstream requirements: RS_BRF_01704, RS_BRF_01720

⌈

Description:	The AUTOSAR CAN Transport Layer shall be statically configurable to support either single or multiple connections in an optimizing way. This configuration is done Pre-Compile-Time
Rationale:	When an ECU enables gateway capabilities, it must handle different message transmissions concurrently across distinct sub-networks. So the AUTOSAR Transport Layer allows concurrent connections. But, most ECU's will only need single connection for diagnostic, which has to be implemented in an optimizing way.
Use Case:	The use case is to provide both single and multiple connections in an optimizing way to save runtime and code size.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01068] The CAN Transport Layer shall identify each N-SDU with a unique identifier.

Upstream requirements: RS_BRF_01704, RS_BRF_01720

⌈

Description:	The CAN Transport Layer identifies each N-SDU with a unique identifier. So the upper layer can address a N-SDU without any assumption on the addressing mode configuration of the CAN-TP. Furthermore, a symbolic name may be assigned for each N-SDU identifier value to simplify usage of the API
Rationale:	Independence of upper layer with the CAN-TP configuration.
Use Case:	The PDU-Router can manipulate all N-SDUs (FlexRay, CAN and LIN) regardless addressing mode particularity of its underlying protocols.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01069] CAN address information and N-SDU identifier mapping

Upstream requirements: RS_BRF_01704, RS_BRF_01720

⌈

Description:	An N-SDU represents either a specific connection defined by a set of address information (N_AI, consisting of MType, N_TAtype, N_TA, N_SA, and N_AE) or a generic connection that represents a dedicated communication path to or from the upper layer for the possible combinations of address information excluding MType and N_TAtype, which are always defined for a connection. Thus, for a specific connection there exists a 1:1 relation between N-SDU ID and address information, while a generic connection is just restricted tocertain addressing formats and functional/physical requests, and possibly to a certain local address.
Rationale:	An N-SDU identifier is used to transmit or receive only one kind of applicative message. N-SDUs are either associated with only one CAN address information (specific connection) or with a set of address information (generic connection). On the other hand, CAN address information is either linked to just one specific connection or to a number of identical, generic connections.
Use Case:	• To transmit or receive an applicative message, the CAN Transport Layer only needs the data and the N-SDU identifier. • To receive and transmit diagnostic messages from different testers, the CAN Transport Layer shall handle the CAN ID directly, using dynamic TX and RX handles of the CAN Interface. • Partitioning of functions among multiple ECUs, therefore an ECU can belong to different functional groups.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01071] The CAN Transport Layer shall identify each N-PDU (also called L-SDU) with a unique identifier

Upstream requirements: RS_BRF_01704, RS_BRF_01720

⌈

Description:	The CAN Transport Layer identifies each N-PDU with a unique identifier. Because the CAN-TP uses the CAN Interface for transmission and reception of N-PDU, these handles shall be unique in both layers. So some common configuration check is needed. Furthermore, a symbolic name may be assigned for each identifier value to simplify the implementation
Rationale:	Each CAN identifier correspond to only one N-PDU identifier of the CAN Transport Layer. So a N-PDU may be completely identified by an identifier.
Use Case:	For optimization reasons, the CAN N-PDU identifier may be different to the CAN identifier.

▽

△

Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01073] The CAN Transport Layer shall be statically configured to pad unused bytes of PDU

Upstream requirements: RS_BRF_01704, RS_BRF_01720, RS_BRF_01712

⌈

Description:	The CAN Transport Layer shall be statically configurable per connection whether to pad unused bytes or not. This affects the last Consecutive Frame (CF), Single Frames (SF) and Flow Control (FC). In the case where CAN FD padding is mandatory for DLC values greater than eight when the length of the data to be transmitted is not equal to one of the discrete length values defined in the ISO 11898-1:2014 DLC table, pad bytes will be added. In case of padding the DLC will always be 8 (bytes) for classic CAN or 8, 12, 16, 20, 24, 32, 48, or 64 (bytes) for CAN FD. The DLC check shall run on the used bytes only. If padding is configured or mandatory, the DLC check shall run over all bytes (DLC = 8, 12, 16, 20, 24, 32, 48, or 64)
Rationale:	Fulfill requirements of legislated OBD communication (ISO 15765-4) and let this feature optional for OEM enhanced diagnostics and applicative communication.
Use Case:	For a full compatibility with old ECUs.
Dependencies:	[SRS_Can_01005], [SRS_Can_01086]
Supporting Material:	–

⌋

[SRS_Can_01074] The Transport connection properties shall be statically configured

Upstream requirements: RS_BRF_01704, RS_BRF_01720

⌈

Description:	• Its unique identifier • Communication direction: sender or receiver • Minimum length of the N-SDU • Associated N-PDU identifier • Physical (1 to 1 communication) or functional (1 to n communication) addressing • Addressing modes: refer to [SRS_Can_01078] • In case of an extended addressing mode connection: N_TA and N_SA values
Rationale:	At runtime the CAN TP module must have all the needed information to manage a transport connection.
Use Case:	This information can be used at generation time to check the network configuration with a TP point of view.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01149] The CAN Transport Layer shall support full-duplex communication for TP channels

Upstream requirements: RS_BRF_01704, RS_BRF_01720

⌈

Description:	The CAN Transport Layer shall support full-duplex communication for TP channels. That means the CAN Transport Layer shall be able to manage a reception and a transmission at same time on the same channel.
Rationale:	Save Can Identifiers.
Use Case:	OEM specific non diagnostic applications which do require a full duplex implementation of the CAN transport protocol.
Dependencies:	–
Supporting Material:	–

⌋

6.2.4.2 Initialization

[SRS_Can_01075] The CAN Transport Layer shall implement an interface for initialization

Upstream requirements: RS_BRF_01704, RS_BRF_01720

⌈

Description:	The CAN Transport Layer implements an interface for initialization. This service shall initialize all global variables of the module and set all transport protocol connections in a default state (Idle)
Rationale:	Basic functionality
Use Case:	Set Transport Layer software to a defined state
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01076] The CAN Transport Layer services shall not be operational before initializing the module

Upstream requirements: RS_BRF_01704, RS_BRF_01720

⌈

Description:	Before using the transmission capabilities of the CAN Transport Layer, it shall be initialized. If it is not the case, the services have to return an error and a development error shall be reported
Rationale:	Basic functionality.
Use Case:	To avoid usage of the module without a complete initialization this could cause the transmission of corrupted frames.
Dependencies:	–
Supporting Material:	–

⌋

6.2.4.3 Normal Operation

[SRS_Can_01078] The AUTOSAR CAN Transport Layer shall support the ISO 15765-2 addressing formats

Upstream requirements: RS_BRF_01704, RS_BRF_01720

⌈

Description:	The AUTOSAR CAN Transport Layer shall support the normal, extended, mixed 11 bit, mixed 29 bit and normal fixed addressing formats of ISO 15765-2.
Rationale:	Basic functionality.
Use Case:	In addition to the normal and extended addressing format, the mixed addressing mode is required for remote diagnostics in automotive area.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01079] The CAN Transport Layer shall be compliant with the CAN Interface module notifications

Upstream requirements: RS_BRF_01704, RS_BRF_01720

⌈

Description:	The CAN Transport Layer shall only implement the CAN Interface notification services concerning TP messages: • Reception notification • Tx confirmation Hint: BusOff management is handled by the CAN State Manager
Rationale:	In AUTOSAR architecture, the CAN Transport Layer is placed between the PDU Router and the CAN Interface.
Use Case:	The CAN Transport Layer has to support the notification services called by the CAN Interface.
Dependencies:	[SRS_Can_01003], [SRS_Can_01009]
Supporting Material:	–

⌋

[SRS_Can_01081] The value of CAN Transport protocol timeouts shall be statically configurable for each connection

Upstream requirements: RS_BRF_01704, RS_BRF_01600, RS_BRF_01720

⌈

Description:	All the defined timeout of the ISO 15765-2 specification are statically configurable for each connection The configuration parameters shall be allowed to be of types Pre-Compile-Time, Link-Time or Post-Build-Time
Rationale:	To adapt the timeout value to the ECU application domain.
Use Case:	The communication constraints may be totally different between a diagnostics connection and an applicative one (e.g. display data).
Dependencies:	–
Supporting Material:	ISO 15765-2[9] specification

⌋

[SRS_Can_01082] Error handling

Upstream requirements: RS_BRF_01704, RS_BRF_02168, RS_BRF_01600, RS_BRF_01720

⌈

Description:	If an unexpected N-PDU is received by the CAN Transport Layer, it shall respect the behavior defined in chapter "unexpected arrival of network protocol data unit" of the ISO-15765-2 specification. For others errors, the CAN-TP just aborts the segmentation session
Rationale:	To define the layer behavior on error.
Use Case:	What happens when receiving the third CF frame instead of the second one?
Dependencies:	–
Supporting Material:	ISO 15765-2[9] specification

⌋

[SRS_Can_01086] Data padding value of unused bytes

Upstream requirements: RS_BRF_01720

⌈

Description:	When the CAN Transport Layer is configured to have fixed data length (DLC = 8), the PDUs are sent without initializing the unused bytes
Rationale:	Setting unused data in the last frame to a specific value will result in increased runtime and resources needs within the μC.

▽

△

Use Case:	The ISO 15765-4 recommendation for OBD communication explicitly says that CAN DLC contained in every diagnostic CAN frame shall always be set to eight and that unused data bytes of a CAN frame are undefined.
Dependencies:	–
Supporting Material:	ISO 15765-4[10] §7

⌋

[SRS_Can_01116] The AUTOSAR CAN Transport Layer shall be able to manage both normal and extended modes in parallel

Upstream requirements: RS_BRF_01720

⌈

Description:	When the CAN Transport Layer is configured to support more than one connection, it should also be possible to configure if it has to deal with both normal and extended addressing mode in parallel or only one of the normal or extended addressing mode
Rationale:	Do not constrain communication capabilities when concurrent connection is allowed. But let it as an OEM specific decision.
Use Case:	A CAN sub-network could mix connection with either normal or extended addressing mode e.g. usage of OBD (normal addressing) and UDS (extended addressing) in parallel
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01148] The AUTOSAR CAN Transport Layer shall provide a service to enable dynamic setting of protocol parameters

Upstream requirements: RS_BRF_01720

⌈

Description:	The AUTOSAR CAN Transport Layer shall provide a service to change BS and STmin parameters during run-time.This service enable the dynamic setting of protocol parameters according to ISO 15765-2 specification.
Rationale:	Dynamic slow down of communication.
Use Case:	Slow down a flash reprogramming process in case high performance ECUs are connected to networks with less performance gateways. Modify the parameters in case a CAN stack is not post build configurable.
Dependencies:	–

▽

△

Supporting Material:	ISO 15765-2[9] specification

⌋

[SRS_Can_01163] The AUTOSAR CAN Transport Layer shall support classic CAN and CAN FD communication as specified by ISO 15765-2

Upstream requirements: RS_BRF_01712

⌈

Description:	The CAN Transport Layer shall support classic CAN and CAN FD communication. This includes the support of N-PDUs up to 64 Bytes length, the extension of the maximum transmission length to 4GBytes, and the differentiation between classic CAN and CAN FD communication.
Rationale:	CAN FD capable transport protocol
Use Case:	Utilizing the extended payload and the increased baud rate of CAN FD improves communication performance.
Dependencies:	[SRS_Can_01161], [SRS_Can_01162]
Supporting Material:	ISO 15765-2[9] specification

⌋

6.2.5 CAN Bus Transceiver Driver

6.2.5.1 Configuration

[SRS_Can_01090] The bus transceiver driver package shall offer configuration parameters that are needed to configure the driver for a given bus and the supported notifications

Upstream requirements: RS_BRF_01704, RS_BRF_01136

⌈

Description:	Typical parameters are: • Max. supported baudrate of each bus to enable the detection of configuration errors • Wakeup by bus • Transceiver control via SPI or port pin • Call context of the notification functions (ISR, polling) to enable detection of necessary data consistency mechanisms during configuration time Please refer to the corresponding software specification for a more detailed view
Rationale:	Basic functionality for transceiver configuration.
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01091] The CAN bus transceiver driver shall support the configuration for more than one bus

Upstream requirements: RS_BRF_01704

⌈

Description:	The driver shall be able to support multiple CAN busses on the ECU. It must be possible to configure the used transceiver type independently for each bus. This includes also mixed systems with e.g. two CANs using different bus physics. Only Pre-Compile-Time configuration shall be possible Transceiver handling depends strongly on the used device. Therefore each transceiver may need its own implementation within the driver and only known and supported devices could be selected. A general solution for the transceiver driver for all use cases might not be possible. By default each CAN controller is attached to an own bus and needs therefore an own bus transceiver. In some cases more than one CAN controller is attached to the same bus to increase the number of mailboxes. Two alternatives appear: **a)** These CAN controllers share the same bus transceiver **b)** Each CAN controller has an own bus transceiver Case a) is covered within this spec and shall be supported by this AUTOSAR driver. Case b) is a very rarely used setup and is therefore not covered by this driver
Rationale:	Basic functionality for transceiver configuration
Use Case:	Multi bus systems, e.g. CAN-CAN gateways
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_02002] The CAN bus transceiver driver shall support the configuration for more than one bus

Upstream requirements: RS_BRF_01704

⌈

Description:	The CAN XL transceiver driver shall support the specialties of CAN XL besides CAN 2.0 and CAN FD. Partial Networking shall only be supported with CAN 2.0 frames
Rationale:	Basic functionality for transceiver configuration
Use Case:	Multi bus systems, e.g. CAN-CAN gateways
Dependencies:	–

▽

△

Supporting Material:	–

⌋

[SRS_Can_01095] The bus transceiver driver shall support the compile time configuration of one notification to an upper layer for change notification for "wakeup by bus" events

Upstream requirements: RS_BRF_01712

⌈

Description:	The CAN XL transceiver driver shall support the specialties of CAN XL besides CAN 2.0 and CAN FD. Partial Networking shall only be supported with CAN 2.0 frames
Rationale:	CAN XL provides higher bandwidth than CAN 2.0 or CAN FD and allows native tunneling of Ethernet frames
Use Case:	Bus transceiver state handling for CAN and Ethernet
Dependencies:	Requires an extended CAN Driver that supports CAN XL bus state handling.
Supporting Material:	CiA 611 (see [1])

⌋

[SRS_Can_01154] The bus transceiver driver package shall offer configuration parameters that are required to configure the driver for partial networking

Upstream requirements: RS_BRF_01184, RS_BRF_01088, RS_BRF_01704

⌈

Description:	Typical parameters are: • Partial networking support • CAN ID of the Remote Wake-up Frame (RWUF) • SPI timeout parameter
Rationale:	To support partial networking tranceivers.
Use Case:	Partial network configurations are affected.
Dependencies:	–
Supporting Material:	–

⌋

6.2.5.2 Initialization

[SRS_Can_01096] The bus transceiver driver shall provide an API to initialize the driver internally

Upstream requirements: RS_BRF_01704

⌈

Description:	The driver must be initialized during the power-up/reset sequence of the ECU. Depending on the used drivers to control the transceivers (e.g. DIO, SPI), they must be already available and working when the transceiver driver is initialized. The wakeup reason has to be detected and stored during the execution of the driver initialization, too
Rationale:	Set bus transceivers and driver in a pre-defined and known state
Use Case:	Basic functionality for transceiver control.
Dependencies:	[SRS_Can_01103] The bus transceiver driver setup information must provide the necessary configuration data to enable the generation tool to select the appropriate control mechanism (e.g. SPI, I/O ports) and to guarantee the correct allocation of the necessary communication resources and initialization sequences.
Supporting Material:	–

⌋

[SRS_Can_01155] The bus transceiver driver shall support the selection of configuration sets

Upstream requirements: RS_BRF_01704, RS_BRF_01136

⌈

Description:	The CAN Interface shall support the selection of one configuration set out of a list of different static configuration sets. This shall be done by a parameter passed via the initialization interface. This is typically done once during startup
Rationale:	Support of different configurations during runtime
Use Case:	Rationale of this request is that at the startup of the ECU some external condition could determine the ECU configuration, without needing coding through a tester or an EOL process (e.g. a coded connection plug, whichsignals through a digital code were an ECU is connected in a given vehicle, hence determining the necessary configuration)
Dependencies:	[SRS_Can_01096]
Supporting Material:	–

⌋

6.2.5.3 Normal Operation

[SRS_Can_01097] CAN Bus Transceiver driver API shall be synchronous

Upstream requirements: RS_BRF_01704

⌈

Description:	The bus transceiver driver API shall execute the requested action immediately and shall deliver the result state immediately to the caller. This will ease up the implementation of wakeup and sleep concepts within the AUTOSAR BSW stack. Some API may require an asynchronous behaviour due to hardware limitations (SPI).
Rationale:	Better usage of transceiver functionality in the complex AUTOSAR BSW environment.
Use Case:	Atomic transition to other operation mode; easier and better abstraction for upper layers like the ECU state manager or ComManager. Improved testability compared to asynchronous handling.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01098] The bus transceiver driver shall support an API to send the addressed transceiver into its Standby mode

Upstream requirements: RS_BRF_01704

⌈

Description:	Many transceivers support the transition to the Sleep mode only via the transition to Standby mode. In addition, some power concepts have the need to set the transceiver to Standby only instead of Sleep mode. Not all transceivers will support such a state. If this is true for a given device, the driver shall confirm the state transition with success
Rationale:	Implementation of ECU low power modes with wakeup via bus and internal.
Use Case:	The upper service layers agreed together with other nodes to set the bus into the sleep mode. The transceiver shall be switched now to a state where the wakeup via bus is supported and the power consumption is as low as possible for the current state of the ECU.
Dependencies:	[SRS_Can_01099]
Supporting Material:	–

⌋

[SRS_Can_01099] The bus transceiver driver shall support an API to send the addressed transceiver into its Sleep mode

Upstream requirements: RS_BRF_01704

⌈

Description:	The transition to sleep mode will be requested with this API. Not all transceivers will support such a state. If this is true for a given device, the drive shall confirm the state transition with success
Rationale:	Implementation of ECU low power modes with wakeup via bus and internal.
Use Case:	The upper service layers agreed together with other nodes to set the bus into the sleep mode. The transceiver is already in StandBy and shall be switched to Sleep with lowest power consumption. Please note that the state sleep of the transceiver is often similar to the state "unpowered" of the ECU.
Dependencies:	[SRS_Can_01098]
Supporting Material:	–

⌋

[SRS_Can_01100] The bus transceiver driver shall support an API to send the addressed transceiver into its Normal mode

Upstream requirements: RS_BRF_01704

⌈

Description:	All transceivers support this state due to it's the "working state"
Rationale:	Communication!
Use Case:	All communication must be enable to communicate.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01101] The bus transceiver driver shall support an API to read out the current operation mode of the transceiver of a specified bus within the ECU

Upstream requirements: RS_BRF_01704

⌈

Description:	The current operation mode of the transceiver will be necessary for upper layers (e.g. diagnostics). The API shall always return the current state seen by the transceiver driver (this may be a locally stored state, too)
Rationale:	State access to transceiver driver

▽

△

Use Case:	Check for current operational mode during development and via diagnostic command.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01103] The bus transceiver driver shall support an API to read out the reason of the last wakeup of a specified bus within the ECU

Upstream requirements: RS_BRF_01704

⌈

Description:	The transceiver driver shall be able to store the local view "who has requested the wakeup: bus or internally". **Bus:** The bus has caused the wakeup. **Internally:** The wakeup has been caused by software **Sleep:** The transceiver is in operation mode sleep and no wakeup has been occurred. **Partial network wake-up:** If the transceiver hardware supports a Partial network wake-up **Wake pin:** An edge on the wake pin of the transceiver (if present) has caused the wakeup. The wakeup reason should be "sleep" when the operation mode is not Normal and no wakeup has been occurred. When a wakeup has occurred, the API shall always return the first detected wakeup reason (e.g. if a wakeup by bus occurs and than nearly at the same time an internal wakeup, the wakeup reason is "bus".). After leaving the operation mode Normal, the wakeup reason shall be set to "sleep" again
Rationale:	Detection of wakeup reason during development and via diagnostic command. May also be used by the NM or ECU state manager.
Use Case:	–
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01106] The bus transceiver driver shall call the appropriate callback function of EcuM in case a wakeup by bus event is detected

Upstream requirements: RS_BRF_01704, RS_BRF_01064, RS_BRF_01680

⌈

Description:	The CAN Bus Transceiver Driver gets a wake up by bus events either through a notification of a lower layer or through polling lower layers. In these cases bus transceiver driver will call appropriate API of EcuM to hand over the event. It shall be possible to support more than one bus within the ECU with this notification. This requirement only applies for transceivers with the appropriate wakeup capability
Rationale:	Efficient coupling between bus transceiver driver and upper layers.
Use Case:	The bus transceiver detects a wakeup condition on the bus and shows this to the μC via e.g. a port pin. Further handling depends on current ECU state. Assumed the ECU is halted, the change on the port may terminate the HALT statement and let the processor continue its work. The assigned port interrupt will be executed and this handler is called. Now, the transceiver driver will store the wakeup reason and give the call via this notification to e.g. the NM to let the NM decide how to handle the event. See [SRS_Can_01095] for details, too.
Dependencies:	Upper layer, i.e. one of (bus specific) NM or ECU state manager. [SRS_Can_01095], [SRS_Can_01138]
Supporting Material:	–

⌋

[SRS_Can_01138] The CAN Bus Transceiver Driver shall provide one callback function for lower layer ICU Driver for wake up by bus events

Upstream requirements: RS_BRF_01704, RS_BRF_01064, RS_BRF_01680

⌈

Description:	ICU driver shall call this API in case of wake up by bus events. One parameter of this function shall refer to the CAN bus which has caused the wakeup by bus event. This API shall be compile time configurable and only available if the corresponding bus transceiver has wakeup capability. If support of wake up by bus is disabled or wake up by bus events are polled this functions shall be removed. This API shall be synchronous or asynchronous depending on the transceiver communication.
Rationale:	Efficient coupling between lower layers and bus transceiver driver.
Use Case:	Notification of wake up by bus events by lower layer.
Dependencies:	[SRS_Can_01106]

▽

△

Supporting Material:	–

⌋

[SRS_Can_01156] The bus transceiver driver shall support wake up events by a Remote Wake-up Pattern (RWUP) or Remote Wake-up Frame (RWUF) if partial networking is supported by the tranceiver hardware

Upstream requirements: RS_BRF_01704, RS_BRF_01104, RS_BRF_01680, RS_BRF_01664

⌈

Description:	If partial networking is supported by bus tranceiver hardware, then the wake-up reasons Wake-up Pattern (RWUP) or Remote Wake-up Frame (RWUF) shall be supported by the bus transceiver driver.
Rationale:	Additional wake-up reasons for partial networking transceivers
Use Case:	Partial network configurations are affected.
Dependencies:	[SRS_Can_01106]
Supporting Material:	–

⌋

[SRS_Can_01107] The CAN Transceiver Driver shall support the situation where a wakeup by bus occurs during the same time the transition to standby/sleep is in progress

Upstream requirements: RS_BRF_01704, RS_BRF_01680, RS_BRF_01664

⌈

Description:	Wakeup by bus is always asynchronous to the internal transition to sleep. In worst case, the wakeup occurs during the transition to sleep. This situation must be covered by the software design and explicitly tested for each ECU. The driver shall create a wakeup notification by bus immediately after the API to enter the standby/sleep mode has finished. The calling/controlling component (NM or ECU state manager) must be capable to handle the wakeup immediately after requesting the standby/sleep
Rationale:	Safe wakeup and sleep handling.
Use Case:	All busses with a wakeup by bus are affected.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01115] The bus transceiver driver shall support an API to enable and disable the wakeup notification for each bus separately

Upstream requirements: RS_BRF_01704, RS_BRF_01680, RS_BRF_01664

⌈

Description:	To enable upper layers to command the bus transceiver safe into its standby and/or sleep state, an additional API to disable and enable the wakeup notification is necessary. If the notification is disabled, driver shall not perform the notification but store the event internally until the notification is enabled again. The notification shall then be processed immediately. It shall be possible to clear a pending wakeup event. If no further wakeup event occurs, no notification shall be performed after enabling the notification again. If a further wakeup event occurs it shall be notified
Rationale:	Safe wakeup and sleep handling.
Use Case:	All busses with a wakeup by bus are affected.
Dependencies:	–
Supporting Material:	–

⌋

6.2.5.4 Shutdown Operation

[SRS_Can_01108] The bus transceiver driver shall support the AUTOSAR ECU state manager in a way that a safe system startup and shutdown is possible

Upstream requirements: RS_BRF_01704, RS_BRF_01096

⌈

Description:	In general, for startup the bus transceivers shall not be enabled until the power supply is available and stable to prevent errors on the bus. Also the communication hardware and driver must not be enabled until the transceiver is configured into its normal operation mode. For shutdown, the communication must be stopped according to the AUTOSAR NM algorithm, the CAN/LIN drivers must be stopped and then the transceivers may be set to standby/sleep, too. The correct sequence depends on the used bus and the wakeup sleep concept of AUTOSAR
Rationale:	Safe system start up and shut down
Use Case:	Systems with support for wakeup by bus.
Dependencies:	–
Supporting Material:	See joint work group meeting WP CAN/LIN and WP Mode Management on 2005-01-11/12 for results.

⌋

[SRS_Can_01157] The bus transceiver driver shall provide an API for clearing the WUF bit in the tranceiver hardware

Upstream requirements: RS_BRF_01704, RS_BRF_01680, RS_BRF_01664

⌈

Description:	This API is part of the shutdown flow of a CAN communication channel. The API clears the WUF flag in the transceiver hardware to be able to signal a following wake-up frame. For CAN transceivers supporting Partial Networking the detection of wake-up frames is also possible in transceiver normal mode. This ensures that no wake-up frame is lost during ECU transition to standby mode, after the WUF flag has been cleared.
Rationale:	Safe system start up and shut down
Use Case:	Systems with support for partial networking.
Dependencies:	–
Supporting Material:	–

⌋

6.2.5.5 Fault Operation

[SRS_Can_01109] The bus transceiver driver shall check the control communication to the transceiver and the reaction of the transceiver for correctness

Upstream requirements: RS_BRF_01704, RS_BRF_01544

⌈

Description:	Depending on the supported transceiver device, the driver shall check the correctness of the executed control communication and the operation mode a transceiver is in. A separation of errors according to SRS_BSW_00337 shall be done
Rationale:	Diagnostics and trouble shooting
Use Case:	1. Detection of defect or misbehaving transceiver hardware 2. Detection of corrupted SPI communication The check shall only be applied to errors within the transceiver or the transceiver control communication (ports or SPI), i.e. errors caused by malfunction of the μC, SW or a defect transceiver device. "Errors" caused by the "outer world" (e.g. disturbed bus lines or ground offsets) are not in the scope of this API.
Dependencies:	–
Supporting Material:	–

⌋

6.3 Non-Functional Requirements (Qualities)

6.3.1 CAN Driver

[SRS_Can_01033] The CAN Driver shall fulfill the general requirements for Basic Software Modules as specified in AUTOSAR_SRS_SPAL

Upstream requirements: RS_BRF_01704

⌈

Description:	Based on Requirements in Document AUTOSAR_SRS_SPAL version 2.0.0
Rationale:	Re-use of requirements validfor all Drivers
Use Case:	CAN Driver is in the same layer as other Drivers (SCI, SPI). Therefore the CAN driver shall fulfill the general SPAL requirements also.
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01034] The CAN Driver shall offer a Hardware independent interface.

Upstream requirements: RS_BRF_01704, RS_BRF_01552

⌈

Description:	The Interface between CAN Driver and CAN Interface shall be independent from underlying hardware. The implementation of the CAN Driver is hardware dependent and statically configurable
Rationale:	Portability
Use Case:	Same CAN Interface implementation can be used for different μCs.
Dependencies:	[SRS_Can_01001]
Supporting Material:	–

⌋

[SRS_Can_01035] The CAN Driver shall support multiple CAN controllers of the same CAN hardware unit

Upstream requirements: RS_BRF_01704

⌈

Description:	The CAN Driver shall support multiple CAN controllers inside one CAN Hardware unit. It shall be possible Pre-Compile-Time to de-select an unused CAN Controller
Rationale:	Coverage of hardware capabilities
Use Case:	Devices exist on the market that incorporate several CAN controller in one device.
Dependencies:	[SRS_Can_01053]
Supporting Material:	–

⌋

6.3.2 CAN Interface (Hardware Abstraction)

[SRS_Can_01121] CAN Interface shall be the interface layer between the underlying CAN Driver(s) and CAN transceiver Driver(s) and Upper Layers

Upstream requirements: RS_BRF_01000, RS_BRF_01008, RS_BRF_01016

⌈

Description:	The CAN Interface is the single interface for all upper Layers for CAN operation. The CAN Interface is the single user of the CAN Driver and the CAN Transceiver Driver.
Rationale:	Interfaces and interaction
Use Case:	Different upper layers (as described in AUTOSAR_WP Architecture_SoftwareArchitecture) may access the same CAN Hardware Unit. Also more than one CAN Hardware Unit with their corresponding drivers (internal and external) may exist in one ECU. Users of the CAN Interface may be the PDU Router, CAN Transport Layer, Network Management and CAN State Manager
Dependencies:	–
Supporting Material:	AUTOSAR_WP Architecture_SoftwareArchitecture

⌋

[SRS_Can_01001] The CAN Interface implementation and interface shall be independent from underlying CAN Controller and CAN Transceiver

Upstream requirements: RS_BRF_01000, RS_BRF_01552

⌈

Description:	The implementation may depend on the amount of available resources of the underlying hardware (i.e. number of CAN Controllers, Hardware Object Handles, HW cancellation allowed) but the Hardware Abstraction Layer encapsulates different mechanisms of hardware access.
Rationale:	Portability and reusability.
Use Case:	Encapsulate implementation details of a specific CAN controller from higher software layers.
Dependencies:	[SRS_Can_01034]
Supporting Material:	–

⌋

6.3.3 CAN State Manager

[SRS_Can_01142] The CAN State Manager shall offer a network abstract API to upper layer

Upstream requirements: RS_BRF_01056

⌈

Description:	The interface of CAN State Manager to the upper layer (ComM) shall be a network abstract interface. The CAN State Manager shall handle the states of peripherals assigned to a network. It shall perform following actions to control the states of the peripherals CAN controller(s) and CAN Transceiver(s): • Init • Start • Stop • WakeUp • Sleep • BusOff Recovery
Rationale:	Abstraction between Com Manager and networks
Use Case:	The bus state manager controls the states of the network specific peripherals of each network.
Dependencies:	–

▽

△

Supporting Material:	–

⌋

[SRS_Can_01014] The CAN State Manager shall offer a network configuration independent interface for upper layers

Upstream requirements: RS_BRF_01064

⌈

Description:	The interface of the CAN State Manager to upper layers shall be independent from the network configuration.
Rationale:	Layer Concept. Information hiding.
Use Case:	Encapsulation of hardware dependencies within CAN Driver and Interface. Modules accessing the CAN State Manager don't need to be hardware specific
Dependencies:	–
Supporting Material:	–

⌋

6.3.4 Transport Layer CAN

[SRS_Can_01065] The AUTOSAR CAN Transport Layer shall be based on ISO 15765-2 and 15765-4 specifications

Upstream requirements: RS_BRF_01720

⌈

Description:	If no requirement is explicitly added or excluded, the implementation of the AUTOSAR CAN Transport Layer shall follow the ISO 15765-2 specification for OEM enhanced (diagnostics or applicative) communication and ISO 15765-4 for on-board diagnostics (OBD) communication
Rationale:	Reuse of existing standards for AUTOSAR BSW. The ISO 15765-2 and 15765-4 specifications are the most used CAN Transport Layer in automotive area.

▽

△

Use Case:	Transport protocol on CAN according to ISO 15765-2: • Segmentation of data in transmit direction • Collection of data in receive direction • Control of data flow • Detection of errors (message loss/doubling/sequence) The network layer described in ISO 15765-4 specification is in accordance with ISO 15765-2 with some restrictions/additions. Refer to the AUTOSAR CAN Transport Protocol software specification for the appropriate version
Dependencies:	–
Supporting Material:	ISO 15765-2[9] and ISO 15765-4[10] specifications

⌋

[SRS_Can_01111] The CAN Transport Layer shall be the interface layer between PDU Router and CAN Interface for CAN messages needing transport protocol functionalities

Status: OBSOLETE
Use instead: SRS_Can_02007
Upstream requirements: RS_BRF_01720

⌈

Description:	The CAN Transport Layer is used by the PDU Router to transmit and receive CAN messages coming from the Diagnostic Communication Manager. Because the PDU Router communicates through both CAN Transport and CAN Interface, their two interfaces shall be coherent (i.e. if they provide a similar primitive, for example Transmit, parameters of those primitives must be as similar as possible). To process transmission the CAN Transport module uses services of the CAN Interface
Rationale:	Interfaces and interaction
Use Case:	By using coherent API (homogeneity of service parameters and so on) the readability and maintainability of source code are improved.
Dependencies:	BSW01118
Supporting Material:	AUTOSAR_WP Architecture_SoftwareArchitecture

⌋

[SRS_Can_02007] The CAN Transport Layer shall be the interface layer between PDU Router and L-SDU Router for CAN messages needing transport protocol functionalities

Status: DRAFT
Upstream requirements: RS_BRF_01720

⌈

Description:	The CAN Transport Layer is used by the PDU Router to transmit and receive CAN messages coming from the Diagnostic Communication Manager. The PDU Router communicates through CAN Transport Layer and L-SDU Router with the CAN interface. Therefore two interfaces shall be coherent (i.e. if they provide a similar primitive, for example Transmit, parameters of those primitives must be as similar as possible). To process transmission the CAN Transport module uses services L-SDU Router, which forward the calls to the CAN Interface
Rationale:	Interfaces and interaction
Use Case:	By using coherent API (homogeneity of service parameters and so on) the readability and maintainability of source code are improved.
Dependencies:	BSW01118
Supporting Material:	AUTOSAR_WP Architecture_SoftwareArchitecture

⌋

[SRS_Can_01112] The CAN Transport Layer interface shall be independent of its internal communication configuration

Upstream requirements: RS_BRF_01720

⌈

Description:	The CAN Transport Layer shall offer the PDU Router an interface that is completely independent to its internal communication configuration (N_TA value, extended or normal addressing mode, functional or physical addressing, etc.) and implementation. The interface shall just deal with PDU identifiers and data units (N-SDU) properties
Rationale:	Layered Software Architecture. Information hiding. Common interface for all applications
Use Case:	–
Dependencies:	[SRS_Can_01014]
Supporting Material:	–

⌋

6.3.5 CAN Bus Transceiver Driver

6.3.5.1 Timing Requirements

[SRS_Can_01110] CAN Bus Transceiver driver shall handle the transceiver specific timing requirements internally

Upstream requirements: RS_BRF_01704

⌈

Description:	The communication between the μC and the transceiver is performed via ports or SPI or both. If ports are used, applying values in a predefined sequence and with a given timing to the ports are used to communicate and change the hardware operation modes. These sequences and timings must be handled within the bus transceiver driver. Small times like the 50μs for TJA1054 "reaction time of go-to-sleep command" may be implemented as a wait loop inside the driver. Disadvantages are that this time is lost for the other software and the wait time depends on the used μC and e.g. system clock. Large wait times (e.g. >200μs) may require an asynchronous API of the bus transceiver driver. Disadvantage is then that the complete API and usage will be different for such a hardware device
Rationale:	Correct handling of used transceiver
Use Case:	E.g. toggling a port pin performs the transition from StandBy to Sleep for the TJA1054. The port value must be kept for at least 50μs to guarantee the transceiver has detected and handled the request in hardware.
Dependencies:	–
Supporting Material:	–

⌋

6.3.6 CAN Driver and Interface together

This chapter describes requirements that shall be fulfilled by the CAN Driver and CAN Interface together.

[SRS_Can_01125] The CAN stack shall ensure not to lose messages in receive direction

Upstream requirements: RS_BRF_01704

⌈

Description:	The CAN stack shall ensure that the HW receive buffer is read out in a time frame that no message is lost for a bus load of 100% with a payload of 1 byte
Rationale:	It shall be possible to work with message bursts without loss of data. This requirement intentionally uses CAN frames with 1 byte payload. They produce more overhead to process them than longer ones. 0 byte messages are seldom used. Hint: Of course this doesn't imply that the general usage of 0 Byte messages is forbidden
Use Case:	See rationale
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01126] The CAN stack shall be able to produce 100% bus load

Upstream requirements: RS_BRF_01704

⌈

Description:	The CAN stack shall be able to produce 100% bus load (except gaps resulting due to not using multiplexed HW transmit buffers). This requirement intentionally uses CAN frames with 1 byte payload. They produce more overhead to process them than longer ones. 0 byte messages are seldom used. Hint: Of course this doesn't imply that the general usage of 0 Byte messages is forbidden
Rationale:	Service the maximum speed of the used CAN bus.
Use Case:	See rationale
Dependencies:	–
Supporting Material:	–

⌋

[SRS_Can_01139] The CAN Interface and Driver shall offer a CAN Controller specific interface for initialization

Upstream requirements: RS_BRF_01136, RS_BRF_01704

⌈

Description:	This service shall initialize the CAN Controller specific configuration like e.g. parameters concerning Baud Rate [SRS_Can_01038]. This service is typically used for re-initialization after e.g. BusOff, but not explicitly restricted to that case. This function call shall only return without error if the CAN driver's state machine is in STOPPED mode. The selection of one out of several configuration sets shall be supported by passing a parameter with the API
Rationale:	Basic functionality.
Use Case:	–
Dependencies:	See description
Supporting Material:	–

⌋

7 References

[1] Specification of CAN Driver
AUTOSAR_CP_SWS_CANDriver

[2] Specification of CAN Interface
AUTOSAR_CP_SWS_CANInterface

[3] Specification of CAN State Manager
AUTOSAR_CP_SWS_CANStateManager

[4] Specification of CAN Transport Layer
AUTOSAR_CP_SWS_CANTransportLayer

[5] Specification of CAN Transceiver Driver
AUTOSAR_CP_SWS_CANTransceiverDriver

[6] Standardization Template
AUTOSAR_FO_TPS_StandardizationTemplate

[7] ISO 11898-1:2015 – Road vehicles – Controller area network (CAN)

[8] IEEE Standard 1722-2016 - IEEE Standard for a Transport Protocol for Time-Sensitive Applications in Bridged Local Area Networks

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

[10] ISO 15765-4 – Diagnostics on controller area network (CAN) – Part 4: Requirements for emission-related systems (Release 2005 01-04)

A Change history of AUTOSAR traceable items

A.1 Traceable item history of this document according to AUTOSAR Release R24-11

A.1.1 Added Requirements in R24-11

[SRS_Can_02004] [SRS_Can_02005] [SRS_Can_02006] [SRS_Can_02007]

A.1.2 Changed Requirements in R24-11

[SRS_Can_01002] [SRS_Can_01003] [SRS_Can_01045] [SRS_Can_01111] [SRS_-Can_02001] [SRS_Can_02002] [SRS_Can_02003]

A.1.3 Deleted Requirements in R24-11

none