Document Title	Requirements on CAN
Document Owner	AUTOSAR GbR
Document Responsibility	AUTOSAR GbR
Document Version	2.1.2
Document Status	Final
Part of Release	2.1
Revision	0014

Document Change History

Date	Version	Changed by	Change Description
24.01.2007	2.1.2	AUTOSAR Administration	• "Advice for users" revised • "Revision Information" added
18.12.2006	2.1.1	AUTOSAR Administration	• PDF file corrections made
04.12.2006	2.1.0	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 BSW01069 and BSW01074 • Legal disclaimer revised
01.04.2006	2.0.0	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
31.05.2005	1.0.0	AUTOSAR Administration	Initial release

Page left intentionally blank

Disclaimer

Any use of these specifications requires membership within the AUTOSAR Development Partnership or an agreement with the AUTOSAR Development Partnership. The AUTOSAR Development Partnership will not be liable for any use of these specifications.

Following the completion of the development of the AUTOSAR specifications commercial exploitation licenses will be made available to end users by way of written License Agreement only.

Advice to users of AUTOSAR Specification Documents:

AUTOSAR Specification Documents may contain exemplary items (exemplary reference models, "use cases", and/or references to exemplary technical solutions, devices, processes or software).
Any such exemplary items are contained in the Specification Documents for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such Specification Documents, nor any later documentation of AUTOSAR conformance of products actually implementing such exemplary items, imply that intellectual property rights covering such exemplary items are licensed under the same rules as applicable to the AUTOSAR Standard.

Table of Contents

1 Scope of this document

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

- CAN Driver (Can)
- CAN Interface (CanIf)
- CAN Transport Layer (CanTp)
- CAN Bus Transceiver Driver (CanTrcv)

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 Conventions used

In requirements, the following specific semantics are used (taken from Request for Comment RFC 2119 from the Internet Engineering Task Force IETF)

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119. Note that the requirement level of the document in which they are used modifies the force of these words.

- MUST: This word, or the terms "REQUIRED" or "SHALL", mean that the definition is an absolute requirement of the specification.
- MUST NOT: This phrase, or the phrase „SHALL NOT“, means that the definition is an absolute prohibition of the specification.
- SHOULD: This word, or the adjective "RECOMMENDED", mean that there may exist valid reasons in particular circumstances to ignore a particular item, but the full implications must be understood and carefully weighed before choosing a different course.
- SHOULD NOT: This phrase, or the phrase "NOT RECOMMENDED" mean that there may exist valid reasons in particular circumstances when the particular behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.
- MAY: This word, or the adjective „OPTIONAL“, means that an item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because the vendor feels that it enhances the product while another vendor may omit the same item. An implementation, which does not include a particular option, MUST be prepared to interoperate with another implementation, which does include the option, though perhaps with reduced functionality. In the same vein an implementation, which does include a particular option, MUST be prepared to interoperate with another implementation, which does not include the option (except, of course, for the feature the option provides.)

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 abbrevations

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

4 Requirements Specification

4.1 Non functional requirements for CAN Driver and Interface together

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

4.1.1 [BSW01125] Data throughput read direction

Initiator:	BMW
Date:	2005-03-02
Short Description:	The CAN stack shall ensure not to lose messages in receive direction.
Type:	New
Importance:	High
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:	--
Conflicts:	--
Supporting Material:	--

4.1.2 [BSW01126] Data throughput write direction

Initiator:	BMW
Date:	2005-03-02
Short Description:	The CAN stack shall be able to produce 100% bus load.
Type:	New
Importance:	High
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:	--
Conflicts:	--
Supporting Material:	--

4.1.3 [BSW01139] CAN Controller specific Initialization

Initiator:	VW / IAV
Date:	27.02.2006
Short Description:	The CAN Interface and Driver shall offer a CAN Controller specific interface for initialization
Type:	New
Importance:	Medium
Description:	This service shall initialize the CAN Controller specific configuration like e.g. parameters concerning Baud Rate (BSW01038). 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
Conflicts:	--
Supporting Material:	--

4.2 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].

4.2.1 Non-functional requirements

4.2.1.1 [BSW01033] Usage of SPAL General Requirements

Initiator:	CAS
Date:	06.07.2004
Short Description:	The CAN Driver shall fulfill the general requirements for Basic Software Modules as specified in AUTOSAR_SRS_SPAL.
Type:	New
Importance:	Medium
Description:	Based on Requirements in Document AUTOSAR_SRS_SPAL version 2.0.0
Rationale:	Re-use of requirements valid for 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:	AUTOSAR_SRS_SPAL
Conflicts:	General Requirements of SPAL doesn't have a stable state
Supporting Material:	--

4.2.1.2 [BSW01034] Hardware abstraction

Initiator:	SV
Date:	30.06.2004
Short Description:	The CAN Driver shall offer a Hardware independent interface.
Type:	New
Importance:	Medium
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:	BSW01001
Conflicts:	--
Supporting Material:	--

4.2.1.3 [BSW01035] Multiple CAN Controller Support

Initiator:	SV
Date:	30.06.2004
Short Description:	The CAN Driver shall support multiple CAN controllers of the same CAN hardware unit.
Type:	New
Importance:	Medium
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:	BSW01053
Conflicts:	--
Supporting Material:	--

4.2.2 Functional Requirements

4.2.2.1 Configuration

4.2.2.1.1 [BSW01036] CAN Identifier Length (Standard / Extended) Configuration

Initiator:	SV
Date:	30.06.2004
Short Description:	The CAN Driver shall support Standard Identifier and Extended Identifier
Type:	New
Importance:	Medium
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:	BSW01016
Conflicts:	--
Supporting Material:	--

4.2.2.1.2 [BSW01037] Hardware filter configuration

Initiator:	SV
Date:	30.06.2004
Short Description:	The CAN driver shall allow the static configuration of the hardware reception filter
Type:	New
Importance:	Medium
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:	BSW01018
Conflicts:	--
Supporting Material:	--

4.2.2.1.3 [BSW01038] Bit Timing Configuration

Initiator:	CAS
Date:	06.07.2004
Short Description:	The bit timing of each CAN Controller shall be configurable
Type:	New
Importance:	Medium
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:	BSW01139
Conflicts:	--
Supporting Material:	--

4.2.2.1.4 [BSW01039] CAN Hardware Object Handle definitions

Initiator:	CAS
Date:	06.07.2004
Short Description:	Hardware Object Handles shall be provided for the CAN Interface in the static configuration file.
Type:	New
Importance:	Medium
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:	BSW01016
Conflicts:	--
Supporting Material:	--

4.2.2.1.5 [BSW01040] HW Transmit Cancellation configuration

Initiator:	CAS
Date:	06.07.2004
Short Description:	The CAN driver shall allow the static enabling or disabling of transmit cancellation
Type:	New
Importance:	Medium
Description:	Hardware transmit cancellation shall be statically enabled or disabled It shall be made public in the configuration file, whether the hardware supports transmit cancellation or not. The configuration parameters shall be allowed to be of type Pre-Compile-Time only
Rationale:	The CAN Driver cancels autonomously if a transmit request with higher priority comes from the CAN Interface. In this case the CAN Interface is notified that the pending transmission was cancelled
Use Case:	--
Dependencies:	BSW01016, BSW01133
Conflicts:	--
Supporting Material:	--

4.2.2.1.6 [BSW01058] Configuration of Multiplexed Transmission

Initiator:	Volcano
Date:	20.07.2004
Short Description:	It shall be configurable whether Multiplex Transmission is used
Type:	New
Importance:	Medium
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:	BSW01134
Conflicts:	--
Supporting Material:	--

4.2.2.1.7 [BSW01062] Configuration of polling mode/interrupt driven mode

Initiator:	SV
Date:	20.07.2004
Short Description:	Each event for each CAN Controller shall be configurable to be detected by polling or by an interrupt
Type:	New
Importance:	Medium
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 a periodically called task. It polls the CAN Controller. The appropriate notifications are called based upon the events that occurred. 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:	--
Conflicts:	--
Supporting Material:	--

4.2.2.1.8 [BSW01135] Configuration of multiple TX Hardware Objects

Initiator:	CAS / VW
Date:	11.10.2005
Short Description:	Configuration of multiple TX Hardware Objects
Type:	New
Importance:	Medium

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:	BSW01058, BSW01049
Conflicts:	--
Supporting Material:	--

4.2.2.2 Initialization

4.2.2.2.1 [BSW01041] CAN Driver Module Initialization

Initiator:	CAS
Date:	06.07.2004
Short Description:	The CAN Driver shall implement an interface for initialization
Type:	New
Importance:	Medium
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:	BSW12057 of SPAL SRS
Conflicts:	--
Supporting Material:	--

4.2.2.2.2 [BSW01042] Selection of static configuration sets

Initiator:	SV
Date:	30.06.2004
Short Description:	The CAN Driver shall support dynamic selection of configuration sets.
Type:	New
Importance:	Medium
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 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:	BSW12062 of SPAL SRS
Conflicts:	--
Supporting Material:	--

4.2.2.3 Normal Operation

4.2.2.3.1 [BSW01043] Enable/disable CAN interrupts

Initiator:	SV
Date:	30.06.2004
Short Description:	The CAN Driver shall provide a service to en-/disable interrupts of the CAN Controller.
Type:	New
Importance:	Medium
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:	--
Conflicts:	--
Supporting Material:	--

4.2.2.3.2 [BSW01059] Data Consistency of received L-PDUs

Initiator:	Vector
Date:	20.07.2004
Short Description:	The CAN Driver shall guarantee data consistency of received L-PDUs
Type:	New
Importance:	Medium
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:	--
Conflicts:	--
Supporting Material:	--

4.2.2.3.3 [BSW01045] Reception Indication Service

Initiator:	SV
Date:	30.06.2004
Short Description:	The CAN Driver shall offer a reception indication service.
Type:	New
Importance:	Medium
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 next upper layer and copies the received data)
Dependencies:	BSW01003
Conflicts:	--
Supporting Material:	--

4.2.2.3.4 [BSW01049] Dynamic transmission request service

Initiator:	SV
Date:	30.06.2004
Short Description:	The CAN Driver shall provide a dynamic transmission request service
Type:	New
Importance:	Medium
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: o Pointer L-SDU source o CAN Identifier o DLC
Rationale:	Basic functionality, CAN Standards coverage
Use Case:	Basic-CAN transmit hardware objects
Dependencies:	BSW01008
Conflicts:	--
Supporting Material:	--

4.2.2.3.5 [BSW01051] Transmission Confirmation

Initiator:	SV

Date:	30.06.2004
Short Description:	The CAN Driver shall provide a transmission confirmation service
Type:	New
Importance:	Medium
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:	BSW01009
Conflicts:	--
Supporting Material:	ISO11898 Section 6.3.3 'Recovery management

4.2.2.3.6 [BSW01053] CAN Controller Mode Select Service

Initiator:	SV
Date:	30.06.2004
Short Description:	The CAN Driver shall provide a service to change the CAN controller mode.
Type:	New
Importance:	Medium
Description:	The CAN Driver shall provide a service to change the mode of the specified CAN controller. The following states shall be supported: o UNINIT – The CAN controller is not configured, typically the registers are in reset state o STOPPED – The CAN controller is configured but does not take part in the CAN communication o STARTED – The CAN controller is up and running o 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 this service
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:	BSW01027
Conflicts:	--
Supporting Material:	--

4.2.2.3.7 [BSW01054] Wake-up Notification

Initiator:	CAS
Date:	06.07.2004
Short Description:	The CAN Driver shall provide a notification for wake-up events
Type:	New

Importance:	Medium
Description:	The CAN driver shall notify the CAN Interface in case of a wake-up interrupt. The notification is done by call of a static callback function implemented inside the CAN Interface. 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 state transition is notified to the CAN Interface. The CAN Interface forwards this notification to the responsible layer (typically the COM Manager)
Dependencies:	BSW01032
Conflicts:	--
Supporting Material:	--

4.2.2.3.8 [BSW01122] Support for wakeup during sleep transition

Initiator:	VW/IAV (Vector)
Date:	09.03.2005
Short Description:	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
Type:	New
Importance:	High
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:	--
Conflicts:	--
Supporting Material:	--

4.2.2.3.9 [BSW01132] Mixed mode for notification detection on CAN HW (Interrupt and Polling)

Initiator:	SV
Date:	4.8.2005
Short Description:	The CAN driver shall be able to detect notification events message object specific by CAN-Interrupt and polling
Type:	New
Importance:	High
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 device drivers, which have strong restrictions to guarantee fixed reaction times and which shall ensure predictable behavior.
Dependencies:	Only possible, if the CAN controller supports message object specific configuration of interrupt usage.
Conflicts:	--
Supporting Material:	--

4.2.2.3.10 [BSW01133] HW Transmit Cancellation support

Initiator:	CAS
Date:	06.07.2004
Short Description:	The CAN driver shall support the HW Transmit Cancellation
Type:	New
Importance:	Medium
Description:	The CAN driver shall support the Cancellation of a pending Transmit Request if HW Transmit Cancellation and Multiplexed Transmission are supported by hardware
Rationale:	The CAN Driver cancels autonomously if a transmit request with higher priority comes from the CAN Interface. In this case the CAN Interface is notified that the pending transmission was cancelled
Use Case:	This requirement is necessary to enable the CAN stack to guarantee message latency (GML). This requirement is only useful for standardization, if there is also a requirement for the entire COM stack, that the overall network description has to be optimized due to ensure GML.
Dependencies:	BSW01040, BSW01134
Conflicts:	--
Supporting Material:	--

4.2.2.3.11 [BSW01134] Multiplexed transmission

Initiator:	CAS / VW
Date:	11.10.2005
Short Description:	The CAN Driver shall support multiplexed transmission
Type:	New
Importance:	Medium
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:	BSW01058
Conflicts:	--
Supporting Material:	--

4.2.2.4 Shutdown Operation

There is no shutdown operation necessary for the CAN Driver. All needed actions are covered by BSW01053 already.

4.2.2.5 Fault Operation

4.2.2.5.1 [BSW01055] Bus-off notification

Initiator:	CAS
Date:	06.07.2004
Short Description:	The CAN Driver shall provide a notification for bus-off state
Type:	New
Importance:	Medium
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. Here typically the COM Manager
Dependencies:	BSW01029
Conflicts:	--
Supporting Material:	--

4.2.2.5.2 [BSW01060] No automatic bus-off recovery

Initiator:	Vector
Date:	20.07.2004
Short Description:	The CAN driver shall not recover from bus-off automatically
Type:	New
Importance:	Medium
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:	--
Conflicts:	--
Supporting Material:	--

4.3 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 [CanIf].

4.3.1 Non-functional requirements

4.3.1.1 [BSW01121] Interfaces of the CAN Interface module

Initiator:	WP1.1.2
Date:	01.09.2004
Short Description:	CAN Interface shall be the interface layer between the underlying CAN Driver(s) and CAN transceiver Driver(s) and Upper Layers.
Type:	New
Importance:	High
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_WP1.1.2_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 COM Manager
Dependencies:	--
Conflicts:	--
Supporting Material:	AUTOSAR_WP1.1.2_SoftwareArchitecture

4.3.1.2 [BSW01014] Network configuration abstraction

Initiator:	SV
Date:	30.06.2004
Short Description:	The CAN Interface shall offer a network configuration independent interface for upper layers
Type:	New
Importance:	High
Description:	The implementation of the CAN Interface is dependent on the network configuration represented by CAN communication matrix. The interface of the CAN Interface 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 Interface don't need to be hardware specific
Dependencies:	--

Conflicts:	--
Supporting Material:	--

4.3.1.3 [BSW01001] Hardware Independence

Initiator:	SV, CAS, DC
Date:	30.06.2004
Short Description:	The CAN Interface implementation and interface shall be independent from underlying CAN Controller and CAN Transceiver.
Type:	New
Importance:	High
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:	- BSW161 (General Requirements on Basic Software Modules) - BSW01034
Conflicts:	--
Supporting Material:	--

4.3.2 Functional Requirements

4.3.2.1 Configuration

4.3.2.1.1 [BSW01015] Network Database Information Import

Initiator:	CAS
Date:	06.07.2004
Short Description:	The CAN Interface configuration shall be able to import information from CAN communication matrix.
Type:	New
Importance:	Medium
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:	--

Conflicts:	--
Supporting Material:	--

4.3.2.1.2 [BSW01016] Interface to CAN Driver configuration

Initiator:	CAS
Date:	06.07.2004
Short Description:	The CAN Interface shall have an interface to the static configuration information of the CAN Driver
Type:	New
Importance:	Medium
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:	BSW01036, BSW01039
Conflicts:	--
Supporting Material:	--

4.3.2.1.3 [BSW01017] Polling/interrupt mode

Initiator:	CAS
Date:	06.07.2004
Short Description:	It shall be Pre-Compile-Time configurable, whether the CAN Interface Main Function is supported for polling purposes
Type:	New
Importance:	Medium
Description:	The CAN Interface shall be configurable in the following two modes: Polling: The CAN Interface represents a periodically called task that polls the CAN status with services provided by the CAN Driver. Depending on the status the appropriate notifications are called. The CAN Interface call period shall be statically configurable. The needed polling rates for each CAN Driver may differ. They shall be configurable as a multiplicity of the CAN Interface call period. This is the smallest period of all CAN Driver Main Functions' calling rates. Interrupt driven: The CAN Interface runs either in interrupt context or in a task that is triggered by events created inside the CAN Driver callback functions. Hint: All CAN Driver Notifications must be enabled for that mode.
Rationale:	--
Use Case:	Basic functionality
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.3.2.1.4 [BSW01018] Software Filter

Initiator:	CAS
Date:	06.07.2004

Short Description:	The CAN Interface shall allow the configuration of its software reception filter Pre-Compile-Time as well as Link-Time and Post-Build-Time
Type:	New
Importance:	Medium
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:	BSW01037, BSW01004, BSW01039
Conflicts:	--
Supporting Material:	--

4.3.2.1.5 [BSW01019] DLC check configuration

Initiator:	CAS
Date:	06.07.2004
Short Description:	It shall be Pre-Compile-Time configurable whether a DLC check is performed or not
Type:	New
Importance:	Medium
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:	--
Conflicts:	--
Supporting Material:	--

4.3.2.1.6 [BSW01020] TX-Buffer configuration

Initiator:	CAS
Date:	06.07.2004
Short Description:	The TX-Buffer shall be statically configurable
Type:	New
Importance:	Medium
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:	BSW01011
Conflicts:	--
Supporting Material:	--

4.3.2.2 Initialization

4.3.2.2.1 [BSW01021] CAN Interface Module Power-On Initialization

Initiator:	CAS
Date:	06.07.2004
Short Description:	The CAN Interface shall implement an interface for initialization.
Type:	New
Importance:	Medium
Description:	The CAN Interface shall implement an interface for initialization. This service shall initialize all module global variables and call the initialization function of the underlying CAN Driver
Rationale:	Basic functionality.
Use Case:	A CAN Interface has static variables that need to be initialized, before the CAN Interface can be used.
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.3.2.2.2 [BSW01022] Dynamic selection of static configuration sets

Initiator:	CAS
Date:	06.07.2004
Short Description:	The CAN Interface shall support the selection of configuration sets.
Type:	New
Importance:	Medium
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:	--
Conflicts:	--
Supporting Material:	--

4.3.2.2.3 [BSW01023] Power-on initialization Sequence

Initiator:	CAS
Date:	06.07.2004
Short Description:	CAN Interface initialization sequence
Type:	New
Importance:	Medium
Description:	The CAN Interface shall be initialized in the following sequence: 1. Initialize global variables 2. Reset flags 3. Call of CAN Driver initialization routine 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:	--
Conflicts:	--
Supporting Material:	--

4.3.2.3 Normal Operation

4.3.2.3.1 [BSW01002] Rx-L-PDU dispatching

Initiator:	SV
Date:	30.06.2004
Short Description:	The CAN Interface shall be responsible for the dispatching of the received PDUs.
Type:	New
Importance:	Medium
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:	--
Conflicts:	--
Supporting Material:	--

4.3.2.3.2 [BSW01003] Reception indication dispatcher

Initiator:	SV
Date:	30.06.2004
Short Description:	The appropriate higher communication stack shall be notified by the CAN Interface about an occurred reception.
Type:	New
Importance:	Medium
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:	BSW01045
Conflicts:	--
Supporting Material:	--

4.3.2.3.3 [BSW01114] Data Consistency of transmit L-PDUs

Initiator:	Review WP1.1.2
Date:	05.10.2004

Short Description:	Data Consistency of L-PDUs to transmit shall be guaranteed
Type:	New
Importance:	Medium
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:	--
Conflicts:	--
Supporting Material:	--

4.3.2.3.4 [BSW01004] Software filtering for L-PDU reception

Initiator:	SV
Date:	30.06.2004
Short Description:	Software filtering shall be implemented by the CAN Interface.
Type:	New
Importance:	Medium
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:	BSW01015, BSW01018, BSW01037, BSW01039
Conflicts:	--
Supporting Material:	--

4.3.2.3.5 [BSW01005] DLC check for L-PDU reception

Initiator:	SV
Date:	30.06.2004
Short Description:	The CAN Interface shall perform a check for correct DLC of received PDUs
Type:	New
Importance:	Medium
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:	BSW01015
Conflicts:	--
Supporting Material:	--

4.3.2.3.6 [BSW01006] Rx-L-PDU enable/disable service per CAN channel

Initiator:	SV

Date:	30.06.2004
Short Description:	The CAN Interface shall provide a service to enable/disable L-PDU reception per CAN channel.
Type:	New
Importance:	Medium
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 network, 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:	BSW01013
Conflicts:	--
Supporting Material:	--

4.3.2.3.7 [BSW01007] Tx-L-PDU dispatching

Initiator:	SV
Date:	30.06.2004
Short Description:	The CAN Interface shall dispatch the transmission request by an upper layer module to the desired CAN controller.
Type:	New
Importance:	Medium
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:	--
Conflicts:	--
Supporting Material:	--

4.3.2.3.8 [BSW01008] Transmission request service

Initiator:	SV
Date:	30.06.2004
Short Description:	The CAN Interface shall provide a transmission request service
Type:	New
Importance:	Medium
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:	BSW01011, BSW01020
Conflicts:	--
Supporting Material:	--

4.3.2.3.9 [BSW01009] Transmission confirmation service

Initiator:	SV
Date:	30.06.2004
Short Description:	The CAN Interface shall provide a transmission confirmation dispatcher
Type:	New
Importance:	Medium
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:	BSW01051
Conflicts:	--
Supporting Material:	--

4.3.2.3.10 [BSW01011] Tx buffering

Initiator:	SV
Date:	30.06.2004
Short Description:	The CAN Interface shall provide a transmit buffer
Type:	New
Importance:	Medium
Description:	The CAN Interface shall provide exactly one buffer for each transmit L-PDU. L-PDUs are stored 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 During Tx confirmation the L-PDU with the highest priority will 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 is stored in an own buffer and older ones are overwritten It shall be Pre-Compile-Time configurable whether the CanIf provides transmit buffers or not
Rationale:	Basic functionality
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:	BSW01020
Conflicts:	--
Supporting Material:	--

4.3.2.3.11 [BSW01013] Tx-L-PDU enable/disable service per CAN channel

Initiator:	SV
Date:	30.06.2004
Short Description:	The CAN Interface shall provide a Tx-L-PDU enable/disable service per

	CAN channel.
Type:	New
Importance:	Medium
Description:	NMs require an additional software service to lock and unlock the transmission of outgoing L-PDUs belonging to one CAN network. This functionality has to be placed in the CAN Interface. Decision by WP1.1.2.
Rationale:	Basic functionality
Use Case:	--
Dependencies:	BSW01006
Conflicts:	--
Supporting Material:	--

4.3.2.3.12 [BSW01027] CAN Controller Mode Select service

Initiator:	CAS
Date:	06.07.2004
Short Description:	The CAN Interface shall provide a service to change the CAN Controller mode.
Type:	New
Importance:	Medium
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:	BSW01053
Conflicts:	--
Supporting Material:	--

4.3.2.3.13 [BSW01028] CAN Controller State Service

Initiator:	CAS
Date:	06.07.2004
Short Description:	The CAN Interface shall provide a service to query the CAN controller state.
Type:	New
Importance:	Medium
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:	--
Conflicts:	--
Supporting Material:	--

4.3.2.3.14 [BSW01032] Wake-up Notification

Initiator:	CAS
Date:	06.07.2004
Short Description:	The CAN Interface shall report a wake-up notification of the CAN Driver.
Type:	New
Importance:	Medium
Description:	When the CAN Interface detects a transition to active state (by CAN Driver notification) a notification call-back function shall be called
Rationale:	Basic functionality
Use Case:	--
Dependencies:	BSW01054
Conflicts:	--
Supporting Material:	--

4.3.2.3.15 [BSW01061] Dynamic TX Handles

Initiator:	SV
Date:	20.07.2004
Short Description:	The CAN Interface shall provide dynamic TX Handles.
Type:	New
Importance:	Medium
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 invalid L-PDU ID table
Use Case:	Dynamically calculated TX IDs. Only ranges of IDs are allowed that are known in the network. Thus there is no impact of the receiving node (no dynamic RX Handles required). Typically used by TP, where the target address is coded within the CAN Identifier. The target address can't be statically defined
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.3.2.3.16 [BSW01130] Receive Status Interface of CAN Interface

Initiator:	SV
Date:	04.08.2005
Short Description:	Polling of notification state message based of CAN Interface
Type:	New
Importance:	High
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:	[BSW 157] Notification mechanisms of modules of Microcontroller Abstraction Layer
Conflicts:	--
Supporting Material:	--

4.3.2.3.17 [BSW01131] Mixed mode of notification and polling mechanism

Initiator:	SV
Date:	4.8.2005
Short Description:	The CAN Interface module shall provide the possibility to have polling and callback notification mechanism in parallel.
Type:	New
Importance:	High
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 device drivers, which have strong restrictions to guarantee fixed reaction times and which shall ensure predictable behavior.
Dependencies:	[BSW 157] Notification mechanisms of modules of Microcontroller Abstraction Layer
Conflicts:	--
Supporting Material:	--

4.3.2.3.18 [BSW01136] Notification of first received CAN message

Initiator:	VW / IAV
Date:	07.11.2005
Short Description:	Notification of first received and valid CAN message
Type:	New
Importance:	High
Description:	The CAN Interface module shall provide the possibility to notify the upper layer in case of the reception of the first valid CAN message after sleep mode
Rationale:	Reduce power consumption
Use Case:	An ECU was woken up by a bus event. If within a certain time no valid CAN message is received, the EcuM assumes an erroneous wakeup e.g.

	caused by a spike on the CAN bus lines and afterwards triggers the ECU to go to sleep again
Dependencies:	BSW09097 of ECU State Manager SRS
Conflicts:	--
Supporting Material:	--

4.3.2.3.19 [BSW01129] Receive Data Interface for CAN Interface

Initiator:	SV
Date:	4.8.2005
Short Description:	The CAN Interface module shall provide a procedural interface to read out data of single CAN messages by upper layers (Polling mechanism)
Type:	New
Importance:	High
Description:	After getting information about new received data (by call get status interface BSW157) 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:	[BSW 157] Notification mechanisms of modules of Microcontroller Abstraction Layer [approved] This requirement is equal to reopen the requirement BSW01026
Conflicts:	--
Supporting Material:	--

4.3.2.3.20 [BSW01140] Support of Standard and Extended Identifiers

Initiator:	VW / IAV
Date:	17.10.2006
Short Description:	The CAN Interface shall support both Standard (11bit) and Extended (29bit) Identifiers
Type:	New
Importance:	High
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:	BSW01141
Conflicts:	--

Supporting Material:	--

4.3.2.3.21 [BSW01141] Support of both Standard and Extended Identifiers on one network (optional feature)

Initiator:	VW / IAV
Date:	17.10.2006
Short Description:	The CAN Interface shall support both Standard (11bit) and Extended (29bit) Identifiers at same time on one network
Type:	New
Importance:	High
Description:	This requirement describes an implementation variant beyond BSW01140: 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, BSW01140 is still valid.
Rationale:	--
Use Case:	Usage of cheap Basic CAN Controllers in CAN networks with both Identifier types
Dependencies:	BSW01036
Conflicts:	--
Supporting Material:	--

4.3.2.4 Shutdown Operation

There is no shutdown operation necessary for the CAN Interface

4.3.2.5 Fault Operation

4.3.2.5.1 [BSW01029] Bus-off notification

Initiator:	CAS
Date:	06.07.2004
Short Description:	The CAN Interface shall report bus-off state of a device to an upper layer
Type:	New
Importance:	Medium
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 Communication Manager ComM
Rationale:	Basic functionality
Use Case:	Any state transition is notified by the CAN Interface. The bus-off notification is typically handled by the Com Manager
Dependencies:	BSW01055
Conflicts:	--
Supporting Material:	--

4.4 Transport Layer CAN

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

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

4.4.1 Non-functional requirements

4.4.1.1 [BSW01065] Usage of ISO 15765-2 and ISO 15765-4 specifications

Initiator:	All
Date:	02.08.2004
Short Description:	The AUTOSAR CAN Transport Layer shall be based on ISO 15765-2 and 15765-4 specifications.
Type:	New
Importance:	High
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:	--
Conflicts:	--
Supporting Material:	ISO 15765-2 and ISO 15765-4 specifications

4.4.1.2 [BSW01111] CAN Transport Layer Interfaces

Initiator:	WP1.1.2
Date:	10.09.2004
Short Description:	The CAN Transport Layer shall be the interface layer between PDU Router and CAN Interface for CAN messages needing transport protocol functionalities.
Type:	New
Importance:	High
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
Conflicts:	--
Supporting Material:	AUTOSAR_WP1.1.2_SoftwareArchitecture

4.4.1.3 [BSW01112] Independent interface

Initiator:	PSA
Date:	10.09.2004
Short Description:	The CAN Transport Layer interface shall be independent of its internal communication configuration.
Type:	New
Importance:	High
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:	[BSW01014]
Conflicts:	--
Supporting Material:	--

4.4.1.4 [BSW01120] Multiple CAN Transport Layer instances

Initiator:	WP1.1.2
Date:	02.12.2004
Short Description:	Multiple CAN Transport Layer instances.
Type:	New
Importance:	High
Description:	It shall be possible to configure the number of instances of the CAN Transport Layer per ECU
Rationale:	The number of CAN Transport instances is variable and independent from physical CAN Channels (0 to n CanTp instances might be mapped to one CAN Channel).
Use Case:	See rationale.
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.4.2 Functional Requirements

4.4.2.1 Configuration

4.4.2.1.1 [BSW01066] Concurrent connection configuration

Initiator:	All
Date:	02.08.2004
Short Description:	Concurrent connection configuration.
Type:	New
Importance:	Medium
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:	--
Conflicts:	--
Supporting Material:	--

4.4.2.1.2 [BSW01068] Unique identifier of N-SDU

Initiator:	PSA
Date:	11.08.2004
Short Description:	The CAN Transport Layer shall identify each N-SDU with a unique identifier.
Type:	New
Importance:	High
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:	--
Conflicts:	--
Supporting Material:	--

4.4.2.1.3 [BSW01069] CAN address information and N-SDU identifier mapping

Initiator:	PSA
Date:	11.08.2004
Short Description:	CAN address information and N-SDU identifier mapping
Type:	New
Importance:	High

Description:	CAN address information and N-SDU identifier shall have a one-to-one driven association. In other words, each CAN address information (the CAN identifier plus, in case of extended addressing mode, the N_TA and N_SA) is statically associated with a unique N-SDU identifier, and vice versa. Of course, the CAN frame and its corresponding N-SDU must have the same direction (Rx or Tx). One ECU shall be able to accept different N_TAs for the same CAN ID
Rationale:	A N-SDU identifier is used to transmit or receive only one kind of applicative message. So a N-SDU identifier is associated with only one CAN address information. And a CAN address information is linked to only one N-SDU identifier.
Use Case:	• To transmit or receive an applicative message, the CAN Transport Layer only needs the data and the N-SDU identifier. • Partitioning of functions among multiple ECUs, therefore an ECU can belong to different functional groups
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.4.2.1.4 [BSW01071] Unique identifier of N-PDU

Initiator:	PSA
Date:	11.08.2004
Short Description:	The CAN Transport Layer shall identify each N-PDU (also called L-SDU) with a unique identifier.
Type:	New
Importance:	High
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:	--
Conflicts:	--
Supporting Material:	--

4.4.2.1.5 [BSW01073] Fixed N-PDU data length

Initiator:	Valeo
Date:	24.08.2004
Short Description:	The CAN Transport Layer shall be statically configured to pad unused bytes of PDU.
Type:	New
Importance:	High
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 case of padding they will always contain 8 bytes (DLC = 8)

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:	[BSW01005] [BSW01086]
Conflicts:	--
Supporting Material:	--

4.4.2.1.6 [BSW01074] Transport connection properties

Initiator:	PSA
Date:	11.08.2004
Short Description:	The Transport connection properties shall be statically configured.
Type:	New
Importance:	High
Description:	The CAN Transport connection configuration shall statically assign properties of each N-SDU: - 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: Normal or extended - 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:	--
Conflicts:	--
Supporting Material:	--

4.4.2.1.7 [BSW01117] Only half-duplex communication is supported

Initiator:	COM Task Force
Date:	02.12.2004
Short Description:	Only half-duplex communication is supported
Type:	New
Importance:	High
Description:	The CAN Transport Layer shall only support half-duplex communication for each connection. The CAN Transport Layer shall be able to manage a reception and a transmission at same time on different connections but not on one connection
Rationale:	If both half and full-duplex is possible in one implementation, the need for resources and runtime will be increased significantly.
Use Case:	Reduce resource and price of ECU.
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.4.2.2 Initialization

4.4.2.2.1 [BSW01075] CAN Transport Layer Initialization

Initiator:	PSA
Date:	11.08.2004
Short Description:	The CAN Transport Layer shall implement an interface for initialization.
Type:	New
Importance:	Medium
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:	--
Conflicts:	--
Supporting Material:	--

4.4.2.2.2 [BSW01076] CAN Transport Layer Availability

Initiator:	PSA
Date:	11.08.2004
Short Description:	The CAN Transport Layer services shall not be operational before initializing the module.
Type:	New
Importance:	Medium
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:	--
Conflicts:	--
Supporting Material:	--

4.4.2.3 Normal Operation

4.4.2.3.1 [BSW01078] Support a subset of ISO 15765-2 addressing modes

Initiator:	All
Date:	02.08.2004
Short Description:	The AUTOSAR CAN Transport Layer shall only support a subset of ISO 15765-2 addressing mode.
Type:	New
Importance:	High
Description:	The AUTOSAR CAN Transport Layer supports only the Normal and Extended addressing modes. Hint: The ISO specification 15765-4(2004-11-09) which describes the legislated on-board diagnostics (OBD) requires the usage of only the

	normal addressing format in the case of 11 bit CAN identifiers and only the normal fixed addressing format in the case of 29 bit CAN identifiers. So this requirement implies that an AUTOSAR ECU cannot provide 29 bit CAN identifiers legislated-OBD communication
Rationale:	Define an addressing mode subset to avoid ambiguous specifications and simplify the CAN transport layer implementation.
Use Case:	These two addressing modes are the most used in automotive area.
Dependencies:	--
Conflicts:	--
Supporting Material:	--

4.4.2.3.2 [BSW01079] Compliance with CAN Interface notifications

Initiator:	All
Date:	02.08.2004
Short Description:	The CAN Transport Layer shall be compliant with the CAN Interface module notifications.
Type:	New
Importance:	High
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 ComManager
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:	Reception notification [BSW01003] Tx confirmation [BSW01009]
Conflicts:	--
Supporting Material:	--

4.4.2.3.3 [BSW01081] Connection specific timeout values

Initiator:	All
Date:	24.08.2004
Short Description:	The value of CAN Transport protocol timeouts shall be statically configurable for each connection.
Type:	New
Importance:	High
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:	--
Conflicts:	--
Supporting Material:	ISO 15765-2 specification

4.4.2.3.4 [BSW01082] Error handling

Initiator:	All
Date:	02.08.2004
Short Description:	Error handling
Type:	New
Importance:	High
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:	--
Conflicts:	--
Supporting Material:	ISO 15765-2 specification

4.4.2.3.5 [BSW01086] Data padding value of unused bytes

Initiator:	Valeo
Date:	24.08.2004
Short Description:	If the CAN Transport Layer is statically configured to have CAN frame with fixed data length (DLC = 8), all unused bytes do not need to have a specific value.
Type:	New
Importance:	High
Description:	When the CAN Transport Layer is configured to have fixed data length, 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:	[BSW01073]
Conflicts:	--
Supporting Material:	ISO 15765-4 §7

4.4.2.3.6 [BSW01116] Usage of different addressing modes in parallel

Initiator:	PSA
Date:	07.10.2004
Short Description:	The AUTOSAR CAN Transport Layer shall be able to manage both normal and extended modes in parallel.
Type:	New
Importance:	Medium
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:	BSW01120
Conflicts:	--
Supporting Material:	--

4.5 CAN Bus Transceiver Driver

4.5.1 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.

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.

4.5.2 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.

4.5.2.1 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.

4.5.2.2 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.

4.5.3 Functional Requirements

4.5.3.1 Configuration

4.5.3.1.1 [BSW01090] Configuration Data for CAN Bus Transceiver

Initiator:	Vector
Date:	30.08.2004
Short Description:	The bus transceiver driver package shall offer configuration parameters that are needed to configure the driver for a given bus and the supported notifications.
Type:	New
Importance:	Medium
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:	ECU configuration description
Conflicts:	--
Supporting Material:	--

4.5.3.1.2 [BSW01091] Support for more than one CAN Bus Transceiver

Initiator:	Vector
Date:	30.08.2004
Short Description:	The CAN bus transceiver driver shall support the configuration for more than one bus
Type:	New
Importance:	High
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:	ECU resource template
Conflicts:	--
Supporting Material:	--

4.5.3.1.3 Configuration of bus operation mode after initialization for each CAN Bus Transceiver

Initiator:	Vector
Date:	30.08.2004
Short Description:	The bus transceiver driver shall support the independent configuration of the bus operation mode for each supported bus.
Type:	New
Importance:	High
Description:	Due to the different startup requirements on a multiple CAN bus ECU, the CAN transceiver driver shall support the independent pre-selection of the bus operation mode to which each transceiver is set during the driver initialization
Rationale:	Basic functionality for transceiver configuration
Use Case:	Multi bus systems
Dependencies:	See needs described for CAN interface, ECU state manager, ComManager and NM.
Conflicts:	--
Supporting Material:	--

4.5.3.1.4 [BSW01095] Configuration “Notification for Wakeup by bus”

Initiator:	Vector
Date:	30.08.2004
Short Description:	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.
Type:	New
Importance:	High
Description:	One wakeup by bus event notification shall be supported to one higher layer. The upper layer shall be configurable during compile time. If a transceiver does not support “wakeup by bus”, this notification is never called for this bus
Rationale:	Efficient coupling between bus transceiver driver and upper layers.
Use Case:	See BSW01106
Dependencies:	Upper layer, i.e. one of (bus specific) NM or ECU state manager. BSW01106
Conflicts:	--
Supporting Material:	--

4.5.3.2 Initialization

4.5.3.2.1 [BSW01096] API to initialize the CAN Bus Transceiver Driver

Initiator:	Vector
Date:	30.08.2004
Short Description:	The bus transceiver driver shall provide an API to initialize the driver internally and set then all attached transceivers in their pre-selected operation modes.
Type:	New
Importance:	High
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:	SPI and DIO driver initialization. [BSW01103] 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.
Conflicts:	--
Supporting Material:	--

4.5.3.3 Normal Operation

4.5.3.3.1 [BSW01097] CAN Bus Transceiver driver API shall be synchronous

Initiator:	Vector
Date:	30.08.2004
Short Description:	The bus transceiver driver API shall be synchronous.
Type:	New
Importance:	High
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
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:	ECU state manager, NM. SPAL in case a transceiver is connected via SPI
Conflicts:	--
Supporting Material:	--

4.5.3.3.2 [BSW01098] API to request operation mode StandBy

Initiator:	Vector
Date:	30.08.2004
Short Description:	The bus transceiver driver shall support an API to send the addressed transceiver into its Standby mode.
Type:	New
Importance:	High
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:	[BSW01099]
Conflicts:	--
Supporting Material:	--

4.5.3.3.3 [BSW01099] API to request operation mode Sleep

Initiator:	Vector
Date:	30.08.2004
Short Description:	The bus transceiver driver shall support an API to send the addressed transceiver into its Sleep mode.
Type:	New
Importance:	High
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:	[BSW01098]
Conflicts:	--
Supporting Material:	--

4.5.3.3.4 [BSW01100] API to request operation mode Normal

Initiator:	Vector
Date:	30.08.2004
Short Description:	The bus transceiver driver shall support an API to send the addressed transceiver into its Normal mode.
Type:	New
Importance:	High

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:	--
Conflicts:	--
Supporting Material:	--

4.5.3.3.5 [BSW01101] API to read out current operation mode

Initiator:	Vector
Date:	30.08.2004
Short Description:	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.
Type:	New
Importance:	High
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:	--
Conflicts:	--
Supporting Material:	--

4.5.3.3.6 [BSW01103] API to read out wakeup reason

Initiator:	Vector
Date:	30.08.2004
Short Description:	The bus transceiver driver shall support an API to read out the reason of the last wakeup of a specified bus within the ECU.
Type:	New
Importance:	High
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. 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:	(Bus specific) NM, diagnostics, ECU state manager
Conflicts:	--
Supporting Material:	--

4.5.3.3.7 [BSW01106] Wakeup by bus notification for upper layer

Initiator:	Vector
Date:	30.08.2004
Short Description:	The bus transceiver driver shall call the appropriate callback function of EcuM in case a wakeup by bus event is detected
Type:	New
Importance:	High
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 [BSW01095] Configuration “Notification for Wakeup by bus” for details, too.
Dependencies:	Upper layer, i.e. one of (bus specific) NM or ECU state manager. BSW01095, BSW01138
Conflicts:	--
Supporting Material:	--

4.5.3.3.8 [BSW01138] Wakeup by bus callback for lower layers

Initiator:	Valeo
Date:	28.11.2005
Short Description:	The CAN Bus Transceiver Driver shall provide one callback function for lower layer ICU Driver for wake up by bus events
Type:	New
Importance:	High
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.
Rationale:	Efficient coupling between lower layers and bus transceiver driver.
Use Case:	Notification of wake up by bus events by lower layer.
Dependencies:	BSW01106
Conflicts:	--
Supporting Material:	--

4.5.3.3.9 [BSW01107] Support for wakeup during sleep transition

Initiator:	Vector
Date:	30.08.2004
Short Description:	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
Type:	New
Importance:	High
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:	--
Conflicts:	--
Supporting Material:	--

4.5.3.3.10 [BSW01115] Support API for enable/disable and clear wakeup events

Initiator:	Vector
Date:	11.10.2004
Short Description:	The bus transceiver driver shall support an API to enable and disable the wakeup notification for each bus separately.
Type:	New
Importance:	High
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:	--
Conflicts:	--
Supporting Material:	--

4.5.3.4 Shutdown Operation

4.5.3.4.1 [BSW01108] Safe system startup and shutdown for CAN Bus Transceiver Driver

Initiator:	Vector
Date:	30.08.2004
Short Description:	The bus transceiver driver shall support the AUTOSAR ECU state manager in a way that a safe system startup and shutdown is possible.
Type:	New
Importance:	High
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:	ECU state manager
Conflicts:	--
Supporting Material:	See joint work group meeting WP4.2.2.1.1 and WP4.2.2.1.9 on 2005-01-11/12 for results.

4.5.3.5 Fault Operation

4.5.3.5.1 [BSW01109] CAN Bus Transceiver Driver must check transceiver control

Initiator:	Vector
Date:	30.08.2004
Short Description:	The bus transceiver driver shall check the control communication to the transceiver and the reaction of the transceiver for correctness.
Type:	New
Importance:	Medium
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 [BSW00337] 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:	[BSW00337], [BSW00338], [BSW00339]
Conflicts:	--
Supporting Material:	--

4.5.4 Non-Functional Requirements (Qualities)

4.5.4.1 Timing Requirements

4.5.4.1.1 [BSW01110 CAN Bus Transceiver driver must handle transceiver specific timing requirements

Initiator:	Vector
Date:	30.08.2004
Short Description:	The bus transceiver driver shall handle the transceiver specific timing requirements internally.
Type:	New
Importance:	High
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:	--
Conflicts:	In case large timings are to be handled, a conflict with the requirement [BSW01097] will occur. The conclusion of the review was to require the transceiver and the driver to be able to switch the operation mode in between a given time (e.g. 200µs). If this is not possible, the transceiver will be marked as “not AUTOSAR compatible” and is therefore not supported by this driver. If more experience with AUTOSAR is available, a “version 2” may support such devices then, too.
Supporting Material:	--

5 References

5.1 Deliverables of AUTOSAR

[Can] Specification of CAN Driver
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_CAN_Driver.pdf

[CanIf] Specification of CAN Interface
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_CAN_Interface.pdf

[CanTp] Specification of CAN Transport Layer
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_CAN_TP.pdf

[CanTrcv] Specification of CAN Transceiver Driver
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SWS_CAN_TransceiverDriver.pdf

[SrsSpal] General Requirements on SPAL
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SRS_SPAL_General.pdf

[SrsGeneral] General Requirements on Basic Software Modules
https://svn.autosar.org/repos/10Releases/
AUTOSAR_SRS_General.pdf

5.2 Related standard and norms

5.2.1 ISO

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

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

ISO 15765-4(2005-01-04), Road vehicles – Diagnostics on Controller Area Networks (CAN) – Part4: Requirements for emissions-related systems

5.3 Related Example Transceiver Data Sheets

See current data sheets for e.g. ST L9669, Freescale MC33389, Philips TJA1054 (CAN LowSpeed), TJA1041 (CAN HighSpeed)