Document Title	Specification of Socket Adaptor
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	416
Document Classification	Standard

Document Version	1.2.0
Document Status	Final
Part of Release	4.0
Revision	3

Document Change History			
Date	**Version**	**Changed by**	**Change Description**
03.11.2011	1.2.0	AUTOSAR Administration	• Rectify inconsistencies in synchronicity and reentrancy • Adjust parameter multiplicity • New traceability mechanism
14.10.2010	1.1.0	AUTOSAR Administration	• ComStack Harmonization. • Allow for Post-Build Configuration • API for IP address change notification • Allow full handling of TCP connections
16.12.2009	1.0.0	AUTOSAR Administration	Initial Release Revision

Disclaimer

Advice for users

AUTOSAR specifications 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 specifications for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such specifications, 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 Introduction and functional overview

This document specifies the functionality, API and the configuration of the AUTOSAR Basic Software module Socket Adaptor (SoAd).

SoAd is the module between the PDU Router and a TCP/IP protocol stack (TCP/IP Stack). The term ‘TCP/IP’ refers to a familiy of protocols, for more detail refer to chapter 7.3.

The PDU Router deploys AUTOSAR I-PDUs onto different communication protocols. The routing through a network system type (e.g. CAN, LIN, FlexRay, or TCP/IP) depends on the I-PDU ID.

The TCP/IP protocol stack is not an AUTOSAR module. Figure 1 depicts the TCP/IP stack as it is intended to fit into the AUTOSAR COM stack. The internal functional structure of the TCP/IP stack is shown schematically for information purposes only, but will neither be mandated nor described in any AUTOSAR document. The main purpose of the SoAd module is to create an interface between the PDU Router and a socket based TCP/IP stack. It will map I-PDU IDs to socket connections and vice versa. SoAd is not a TP, as it does not provide segmentation or flow control, all of these functionalities are expected to be implemented by the TCP/IP stack. However, some segmentation of TCP stream data in the receive direction will be required.

SoAd will detect and handle errors resulting from TCP/IP stack operations.

It is an AUTOSAR decision to base the specification of this basic software module on existing standards. Therefore the AUTOSAR Socket Adaptor specification is based on the BSD socket standard (as specified in [21]), but allows for extensions in order to increase performance where needed.

Furthermore the SoAd is expected to have protocol parser extensions using its interfaces towards the TCP/IP stack and towards the PDU Router.

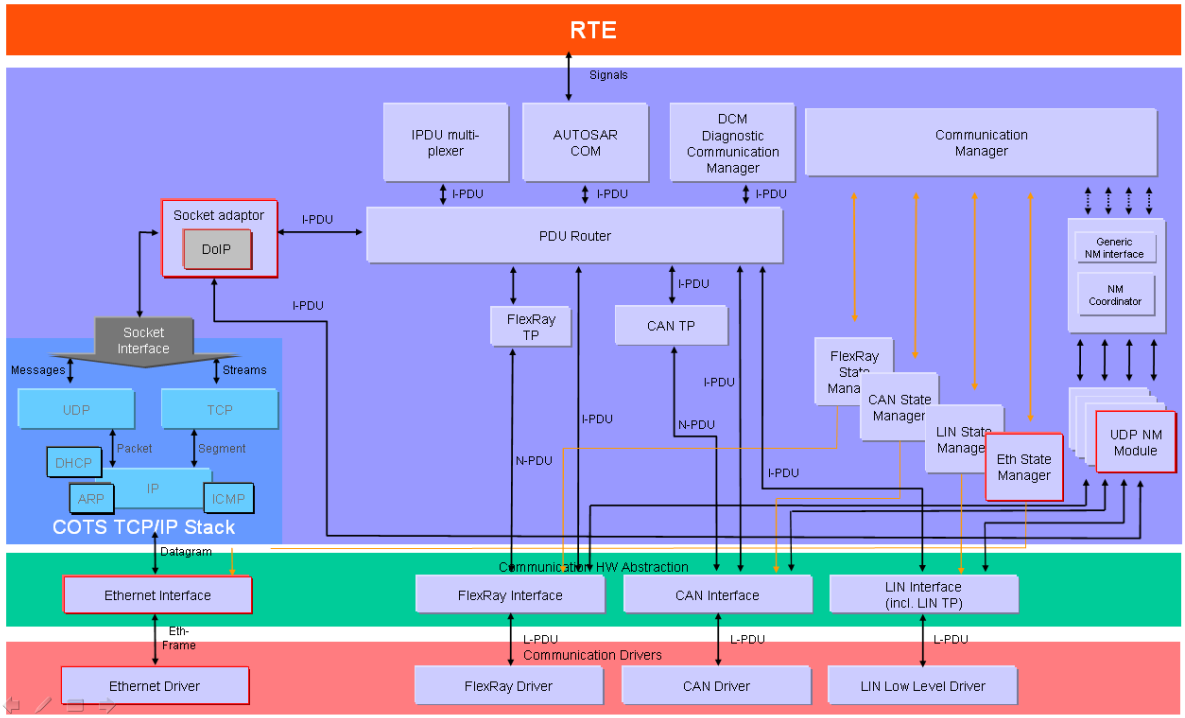


Figure 1: Extended AUTOSAR Communication Stack.

2 Acronyms and abbreviations

Abbreviation / Acronym:	*Description:*
ARP	Address Resolution Protocol
COTS	Commercial Of The Shelf
DEM	Diagnostic Event Manager
DET	Development Error Tracer
DHCP	Dynamic Host Configuration Protocol
DoIP	Diagnostics over IP
HTTP	HyperText Transfer Protocol
IANA	Internet Assigned Numbers Authority
ICMP	Internet Control Message Protocol
IP	Internet Protocol
TCP	Transmission Control Protocol
TCP/IP	A family of communication protocols used in computer networks
TP	Transport Protocol
UDP	User Datagram Protocol
UdpNm	AUTOSAR UDP Network Management

Term:	*Description:*
AUTOSAR Connector	The SoAd serves as a (De)Multiplexer between different PDU sources/suppliers and the TCP/IP stack. The term AUTOSAR connector refers to a source or supplier of a PDU.

3 Related documentation

3.1 Input documents

[1] Layered Software Architecture
AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf

[2] General Requirements on Basic Software Modules
AUTOSAR_SRS_BSWGeneral.pdf

[3] Requirements on Network Management
AUTOSAR_SRS_NetworkManagement.pdf

[4] Specification of CAN Interface
AUTOSAR_SWS_CANInterface.pdf

[5] Specification of Communication Stack Types
AUTOSAR_SWS_CommunicationStackTypes.pdf

[6] Specification of ECU Configuration
AUTOSAR_TPS_ECUConfiguration.pdf

[7] Specification of BSW Scheduler
AUTOSAR_SWS_BSW_Scheduler.pdf

[8] Specification of Communication Manager
AUTOSAR_SWS_ComManager.pdf

[9] Specification of ECU State Manager
AUTOSAR_SWS_ECUStateManager.pdf

[10] Specification of Operating System
AUTOSAR_SWS_OS.pdf

[11] Specification of Diagnostic Event Manager
AUTOSAR_SWS_DiagnosticEventManager.pdf

[12] Specification of Development Error Tracer
AUTOSAR_SWS_DevelopmentErrorTracer.pdf

[13] Specification of Standard Types
AUTOSAR_SWS_StandardTypes.pdf

[14] Specification of Platform Types
AUTOSAR_SWS_PlatformTypes.pdf

[15] Specification of Compiler Abstraction
AUTOSAR_SWS_CompilerAbstraction.pdf

[16] Basic Software Module Description Template
AUTOSAR_TPS_BSWModuleDescriptionTemplate.pdf

[17] Specification of UDP Network Management
AUTOSAR_SWS_UDPNetworkManagement.pdf

[18] Requirements on Ethernet
AUTOSAR_SRS_Ethernet.pdf

[19] List of Basic Software Modules
AUTOSAR_TR_BSWModuleList

3.2 Related standards and norms

[20] IEC 7498-1, “The Basic Model”, IEC Norm, 1994

[21] IEEE Std. 1003.1™, 2004 Edition, “POSIX”
http://www.opengroup.org/onlinepubs/000095399/

[22] ISO Standard on Diagnostics over IP (ISO DoIP), ISO/NP 13400 (to be published)

3.2.1 IETF Requests For Comments (RFCs)

[23] RFC 791, “INTERNET PROTOCOL”

[24] RFC 793, “TRANSMISSION CONTROL PROTOCOL”, (TCP)

[25] RFC 768, “User Datagram Protocol”, (UDP)

[26] RFC 1122, “Requirements for Internet Hosts -- Communication Layers”

[27] RFC 896, "Congestion Control in IP/TCP Internetworks", (Nagle algorithm)

4 Constraints and assumptions

4.1 Limitations

It is intended for this AUTOSAR module to cooperate with a COTS TCP/IP stack not specified as an AUTOSAR SWS. This imposes limitations on the COTS TCP/IP stack as well as on the Socket Adaptor Module (SoAd). While limitations on the SoAd are found as requirements throughout this document, the limitations on the TCP/IP stack are summarized in chapter 5.3.3.

The transmission of data using TCP/IP over Ethernet requires about 60 bytes of header information. This implies that for small messages the header overhead may reach an unacceptably high percentage.

To avoid further protocol overhead, the use of a single socket connection per PDU is described here. However, this solution is very resource intensive, particularly if many small PDUs are to be transmitted. One solution described here as an option is to add a small PDU header, containing an ID and length information. This enables transmission of multiple PDUs via one socket connection. Additionaly a resource conservation scheme is included in this specification as an option.

Furthermore the provisons for further protocol plug-ins are expected to address this issue in later versions.

This document does not cover the assignment of `UDP` or `TCP` port numbers. There is no reserved space within the IANA assigned number range. Each implementer is responsible for managing the used port numbers.

This document does not cover the assignment and management of IP addresses. This might be done dynamically, e.g. by using `DHCP`, or statically. It is the implementers responsibility to prevent address conflicts and achieve compliance with IANA address assignments.

This specification does not prescribe a certain physical layer or data rate.
This specification does not give detailed requirements for the functionality of the TCP/IP stack, but assumes an implementation in line with [26].

This document and the data types therein are intended for use with IPv4 [23], there is no full support for IPv6 in the current version although some provisions are made to include this in later versions.

4.2 Applicability to car domains

N/A

5 Dependencies on other modules

This section outlines relations between the SoAd and other AUTOSAR basic software modules. It contains brief descriptions of the services required by the SoAd from other modules and how other modules will call the SoAd.

5.1 File structure

5.1.1 Code file structure

[SOAD071] ⌈The code file structure shall not be defined within this specification completely. At this point it shall be pointed out that the code-file structure shall include the following files named:

- `SoAd.c` – the source code
- `SoAd_Lcfg.c` – for link time configurable parameters and
- `SoAd_PBcfg.c` – for post build time configurable parameters.
- `SoAd_DoIp.c` – source code of the DoIP protocol plug-in

These files shall contain all link time and post-build time configurable parameters.⌋ ( )

5.1.2 Header file structure

[SOAD072] ⌈The SoAd module shall provide the following H-files:

- `SoAd.h` (for declaration of provided interface functions)
- `SoAd_Cbk.h` (for declaration of provided call-back functions)
- `SoAd_Cfg.h` (for pre-compile time configurable parameters)
- `SoAd_Types.h` (for type defined in SoAd section 8.2) ⌋ ( )

[SOAD073] ⌈The SoAd module shall include the following H-files:

- `TcpIp.h` – header file of the AUTOSAR call-back TCP/IP stack
- `Bsd.h` – header file of the BSD socket TCP/IP stack
- `ComStack_Types.h`
- **Note:** The following header files are indirectly included by `ComStack_Types.h`:
  - `Std_Types.h` (for AUTOSAR standard types)
  - `Platform_Types.h` (for platform specific types)
  - `Compiler.h` (for compiler specific language extensions)
- `Bsd_Types.h` – header file for the BSD socket TCP/IP stack types
- `Det.h` – header file of Det
- `SchM_SoAd.h` (for services of the Basic Software Scheduler)
- `MemMap.h` – header file for Memory Mapping

- `PduR_SoAd.h` (for services of the PDU Router module) ⌋( )

[SOAD070] ⌈The SoAd module shall include the `Dem.h` file. By this inclusion the APIs to report errors as well as the required Event Id symbols are included. This specification defines the name of the Event Id symbols which are provided by XML to the DEM configuration tool. The DEM configuration tool assigns ECU dependent values to the Event Id symbols and publishes the symbols in `Dem_IntErrId.h`. ⌋( )

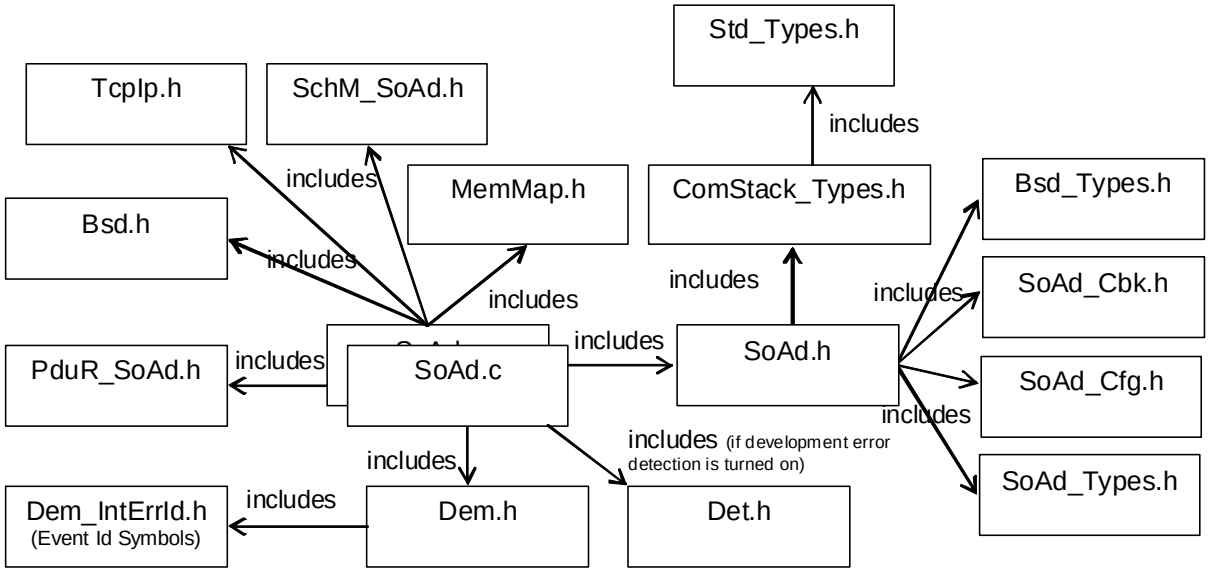


Figure 2: AUTOSAR SoAd header file structure.

5.1.3 Design Rules

[SOAD032] ⌈The code of the Socket Adaptor module (as long as it is written in C) shall conform to the HIS subset of the MISRA C Standard unless specified otherwise. ⌋( )

Rationale: Violations of the MISRA C Standard are permissible only in those cases where compatibility with the COTS TCP/IP Stack BSD Socket interface as specified in IEEE 1003.1 [21] which can not be achieved otherwise.

[SOAD076] ⌈Direct use of compiler and platform specific keywords shall be avoided. Indicate all global data with read-only purposes by explicitly assigning the `const` keyword. ⌋( )

It is allowed to use macros instead of functions where source code is used and runtime is critical.

[SOAD081] ⌈No global data shall be defined in the header files. If global variables have to be used, the definition shall be placed in the source code file. ⌋(BSW00346, BSW158)

Exception: Variables used for debugging.

[SOAD082] ⌈The source code of the SoAd module shall not be processor and compiler dependent.⌋ ( )

[SOAD084] ⌈As the SoAd module is intended to work together with a COTS software component, the data types used there may need to be included, although they do not conform to the AUTOSAR rules for data types. All of these are to be found in `Bsd_Types.h`.⌋ ( )

5.2 AUTOSAR architecture basic concepts

5.2.1 Static configuration

At run-time the Socket Adaptor module must have some basic information required to manage the transport connections.

[SOAD044] ⌈The Socket Connection Table shall be used to collect this information. Some of its entries will be filled or updated at run-time.⌋ (BSW00346, BSW158)

[SOAD043] ⌈The PDU and Socket Routing Tables contain static information only and shall be pre-configured, not to be modified during run-time.⌋ ( )

5.2.1.1 Socket Connection Table

[SOAD018] ⌈The Socket Connection Table contains all relevant information on each socket connection to be operated during run-time.

For each Connection the following parameters are required:

- Socket ID (`uint16`)
  Zero based consecutive connection index (static configuration).
- Socket Handle (`int`)
  (determined at run-time, see section 8.3.4.12 for details)
- Local IP Address (`uint32` for IPv4)
  (static configuration or updated at run-time)
- Local Port (`uint16`)
  (static configuration)
- Remote IP Address (`uint32` for IPv4)
  (static configuration or updated at run-time)
- Remote Port (`uint16`)
  (static configuration or updated at run-time)
- Protocol Type (`SoAd_ProtocolType`)
  TCP or UDP (static configuration).

- TCP Initiate (`boolean`)
  Used for TCP connections only (static configuration)!
- TCP No Delay (`boolean`)
  Used to disable the congestion controll algorithm (static configuration).
- UDP listen only (`boolean`)
  Listen only or allow transmission (static configuration).
- AUTOSAR Connector (`enumeration`)
  PDU Router, DoIP, UdpNm, CDD, or Xcp (static configuration).
- PDU Header enable (`boolean`)
  A header containing ID and length will be added to each PDU transmitted over TCP/IP (static configuration, see 7.3.5 for details).
- Socket API enable (`boolean`)
  BSD or Call-Back (static configuration).
- Socket Resoure Management enable (`boolean`)
  Allows for this socket to be closed when no data is to be sent, in order to preserve resources (static configuration).
- Socket Resoure Management counter (`uint8`)
  Indication of the usage of this socket used by the Socket Resoure Management, if enabled (generated at run-time, see 7.3.4 for details).

All statically configurable items are described in detail in section 10.2.4. ⌋( )

5.2.1.2 PDU Routing Table

[SOAD019] ⌈The PDU Routing Table describes the path for data coming from the PDU Router and gives the socket this data needs to be transmitted on. It is statically configured.

For each PDU it shall contain the following Information:

- Source PDU ID
  PDU ID handed down by the PDU Router.
- Source SDU Length
- Destination Socket ID (`uint16`)
  Refers to the Socket ID in the Socket Connection Table.
- Destination ID (`uint16`)
  The ID sent in the PDU header, if enabled.

Multiple entries for one PDU ID are allowed in this table under the following circumstances:

- If the PDU header is enabled for the socket referred to in Destination Socket ID.

All statically configurable items are described in detail in section 10.2.6. ⌋( )

5.2.1.3 Socket Routing Table

[SOAD020] ⌈The Socket Routing Table describes the path for data received on a socket and gives the PDU ID in which this data needs to be handed to the PDU Router. It is statically configured.

For each Socket it shall contain the following Information:

- Source Socket ID (`uint16`)
  Refers to the Socket ID in the Socket Connection Table.
- Source ID (`uint16`)
  The ID received in the PDU header, if enabled.
- Destination PDU ID
  PDU ID handed up to the PDU Router.
- Destination SDU Length

Multiple entries for one PDU ID are allowed in this table under the following circumstances:

- If the PDU header is enabled for the socket referred to in Source Socket ID.
- On a `UDP` socket listening to multiple source ports, where the source port is used to distinguish PDUs.

All statically configurable items are described in detail in section 10.2.5. ⌋( )

5.3 TCP/IP Software Stack

As mentioned earlier the TCP/IP protocol stack is not defined as an AUTOSAR software module, but is to be added in as a COTS component.

However in some cases additional performance will be required, that is not feasible with off the shelf software. The following chapters describe two different socket based interfaces, a TCP/IP stack might implement either one or both of these.

Some additional requirements on the COTS software to assure compatibility with SoAd are stated in chapter 5.3.3.

5.3.1 BSD Socket API (COTS)

The following services of the TCP/IP stack are called by SoAd if the non-blocking BSD socket API is used:

- `accept`
  By this API service a new TCP socket is created for an incoming connection. This requires an according entry in the Socket Connection Table
- `bind`
  Binds a socket to a local IP number and port.

- `close`
  Closes a socket.
- `connect`
  By this API service a TCP socket is requested to connect to the peer. If the initiating socket is not connection-mode, then connect shall set the sockets peer address, and no connection is made.
- `fcntl`
  Send commands to the TCP/IP stack for configuration.
- `getlasterror`
  Get the error code of the most recent error that occurred.
- `listen`
  By this API service a TCP socket is requested to enter the LISTEN state.
- `poll`
  Requests status information from the TCP/IP stack.
- `recvfrom`
  Receives a datagram and stores the source address.
- `sendto`
  Sends data to a specific destination.
- `setsockopt`
  Set the socket options.
- `socket`
  By this API service a socket handle is requested.

No call-back functions are availabe in the SoAd to be called by the TCP/IP stack, if the BSD socket API is used.

The BSD socket interface may be used for non-blocking calls by any other module.

5.3.2 Call-back Socket API (AUTOSAR)

The following services of the TCP/IP Stack are called by the SoAd if the Call-back socket API is used:

- `TcpIp_Listen`
  By this API service a TCP/UDP socket is requested to enter the LISTEN state.
- `TcpIp_TcpChangeParameter`
  By this API service, parameters of a TCP channel can be changed (e.g. Nagle [27] on/off).
- `TcpIp_TcpClose`
  By this API service a TCP socket is requested to be closed. A FIN segment is sent to the peer [24].
- `TcpIp_TcpConnect`
  By this API service a TCP socket is requested to connect to the peer.
- `TcpIp_TcpReceived`
  By this API service the reception of socket data is confirmed to the TCP/IP stack. The TCP/IP stack assumes that allocated buffers are freed and will increase the advertised TCP window.

- `TcpIp_TransmitTo`
  By this API service, the sending of a TCP/UDP packet is triggered. Depending on configuration of the TCP/IP stack, a TCP segment is sent immediately or after coalescing multiple segments (Nagles algorithm [27]).

The following call-back functions of the SoAd are called by the TCP/IP Stack if the Call-back socket API is used:

- `SoAd_TcpIpTxConfirmation`
  By this API service, the TCP/IP Stack confirms the sending of `TCP/UDP` socket data over the IP network. **Note:** for TCP connections this happens if the data ist ACKed by the peer.
- `SoAd_TcpIpRxIndication`
  By this API service, the TCP/IP Stack indicates the reception of `TCP/UDP` socket data to the SoAd. The SoAd then processes this data.
- `SoAd_TcpAccepted`
  By this API service, the TCP/IP Stack notifies the SoAd that a socket in the LISTEN state became connected.
- `SoAd_TcpConnected`
  By this API service, the TCP/IP Stack notifies the SoAd that a socket for which `TcpIp_TcpConnect()` was called before got actually connected.

[SOAD123] ⌈The Autosar Call-back interface shall only be used for non-blocking calls by the SoAd module.⌋()

Rationale: As the COTS TCP/IP stack has no way of accessing the information in the Socket Connection Table, it can not directly call the module the received data is intended for.

5.3.3 COTS Stack compatibility

IEEE 1003.1 defines standard symbols like `AF_INET`, `POLLWRNORM`, `EAGAIN`, and many others. Unfortunately different implementations may choose different values for these symbolic names. Only TCP/IP stacks implementing the values listed in the SWS SoAd document will be compatible and can be used. The values used here are derived from the FreeBSD implementation.

The ISO standard on DoIP will impose certain requirements on the TCP/IP stack, those are not repeated here, but will have to be considered for implementations using that feature [22].

5.4 PDU Router services

The Socket Adaptor declares and requests certain call-back functions to confirm transmission, confirm transmission cancellation and notify reception of a message from/to the PDU-Router, and request a buffer, to reassemble segmented frames. For

more information about these functions, refer to the PDU Router module specification.

The following call-back functions of the PduRouter are called by the SoAd:

- `PduR_SoAdStartofReception`
  By this API service, the SoAd asks the actual receiver of the message to provide a receive buffer. It is not necessary for the buffer to have at least the same size as the whole received data length (there will be another call in this case) (see 8.6.4.6).
- `PduR_SoAdCopyRxData`
  This API service is called by the SoAd to indicate availability of received data (see 8.6.4.4).
- `PduR_SoAdRxIndication`
  By this API service, the SoAd indicates the completed (un)successful reception of an L-PDU (see 8.6.4.1).
- `PduR_SoAdCopyTxData`
  This API service is called by the SoAd to indicate a request for data to transmit (see 8.6.4.7).
- `PduR_SoAdTxConfirmation`
  By this API service, the SoAd confirms the (un)successful sending of the complete message to the actual sender (see 8.6.4.2).

The following services of the SoAd are called by the PduRouter:

- `SoAdIf_Transmit`
  By this API service, the transmission of a PDU is triggered (see 8.3.3.1).
- `SoAdTp_Transmit`
  By this API service, the sending of socket data is triggered. The SoAd will then ask for a transmit buffer and start sending (see 8.3.3.2).

5.5 UDP NM Module

5.5.1 UDP NM Configurable Interfaces

The following call-back fuctions of the UDP NM are called by the SoAd:

- `UdpNm_SoAdTxConfirmation` (see 8.6.4.2).
- `UdpNm_SoAdRxIndication` (see 8.6.4.1).

5.6 XCP Module

5.6.1 XCP Configurable Interfaces

The following call-back fuctions of the XCP are called by the SoAd:

- `Xcp_SoAdTxConfirmation` (see 8.6.4.2).
- `Xcp_SoAdRxIndication` (see 8.6.4.1).

5.7 DoIP Plug-In

5.7.1 Overview

The term ‘Diagnostics over IP (DoIP)’ refers to an ISO standard describing vehicle diagnostics [22], please refer to chaper 7.2.2 for a use-case description.
For AUTOSAR the functionality of the DoIP plug-in module is to be described in this document at a later time. It will be implemented in the same software module as the SoAd functionality.

[SOAD149] ⌈If a recommendation of ISO WD DoIP [22] is not explicitly excluded in the SWS, the DoIP module shall follow the ISO WD DoIP recommendation. ⌋( )

[SOAD159] ⌈In the `DoIP_Pdu_Routing_Table` each PDU ID is assigned to a DoIP source address and a DoIP target address. ⌋( )

5.7.2 Common Interfaces

[SOAD134] ⌈The DoIP plug-in shall use the same interfaces when calling the PDU Router as the SoAd.

This implies, that DoIP is not able to communicate with e.g. a Complex Device Driver (CDD), as there is no way to configure the <User> in the APIs. ⌋( )

[SOAD135] ⌈The DoIP plug-in shall use the same interfaces when calling the TCP/IP stack as the SoAd (for both socket APIs). ⌋( )

5.7.3 TCP/IP Stack Requirements for DoIP

[SOAD150] ⌈To support DoIP the TCP/IP stack shall implement the network layer requirements stated in [22]. ⌋( )

[SOAD152] ⌈To support DoIP the TCP/IP stack shall implement the transport layer requirements stated in [22]. ⌋( )

[SOAD153] ⌈According to [22] the corresponding number of TCP Scocket Connection Table entries shall be configured (using the specified ports, etc.). ⌋( )

[SOAD154] ⌈To support DoIP the TCP/IP stack shall implement the address assignment mechanisms required by the ISO WD DoIP standard [22]. ⌋( )

[SOAD155] ⌈To support DoIP the TCP/IP stack shall implement the application layer requirements stated in [22]. ⌋( )

[SOAD166] ⌈The <manufacturer specific> part of the “host name option” shall be configurable post built via the `SoAd_DoIPHostNameOpt` parameter.⌋( )

5.7.4 DoIP Socket handling

The DoIP plug-in is in full control of the sockets marked DoIP as the AUTOSAR connector in the Socket Connection Table.

[SOAD128] ⌈The DoIP plug-in shall not act on any of the connections not marked DoIP as connector in the Socket Connection Table.

The DoIP plug-in will use and update the information in the Socket Connection Table just as the SoAd does.⌋( )

[SOAD167] ⌈No resource conservation option shall be used for sockets used for DoIP.⌋( )

[SOAD169] ⌈The SoAd Socket Connection Table shall include the UDP and TCP sockets required by the DoIP plug-in if the DoIp plug/in is in use.⌋( )

[SOAD168] ⌈DoIP connection handling shall be implemented by the DoIP plug-in as specified in [22].⌋( )

5.7.5 Sending of DoIP messages

The DoIP plug-in will use the same funcions to transmit data via the TCP/IP stack as the SoAd does. It may implement either one of the APIs specified by the SoAd.

5.7.6 Reception of DoIP messages

The DoIP plug-in will implement its own strategy to receive messages from the TCP/IP stack. It may implement either one of the APIs specified by the SoAd. If the Call-back Socket API is to be used, configurable call-back interfaces are required to be implemented in the TCP/IP stack.

[SOAD156] ⌈The DoIP plug-in shall process DoIP messages and perform the required actions according to ISO WD DoIP [22].⌋( )

5.8 Other Protocol Handler Modules

It is expected, that other special services like the DoIP plug-in will want to use the TCP/IP stack.

[SOAD200] ⌈These shall not extend the interfaces described here and shall coordinate with the SoAd mainly through the Socket Connection Table.⌋( )

[SOAD201] ⌈They shall be able to use and implement the configurable interfaces.⌋( )

6 Requirements traceability

Requirement	Satisfied by
-	SOAD144
-	SOAD243
-	SOAD039
-	SOAD213
-	SOAD274
-	SOAD227
-	SOAD088
-	SOAD225
-	SOAD264
-	SOAD239
-	SOAD248
-	SOAD167
-	SOAD232
-	SOAD292
-	SOAD146
-	SOAD086
-	SOAD224
-	SOAD028
-	SOAD242
-	SOAD291
-	SOAD184
-	SOAD106
-	SOAD217
-	SOAD154
-	SOAD092
-	SOAD111
-	SOAD059
-	SOAD159
-	SOAD104
-	SOAD177
-	SOAD215
-	SOAD084
-	SOAD195
-	SOAD265
-	SOAD042
-	SOAD157
-	SOAD267
-	SOAD136

-	SOAD036
-	SOAD187
-	SOAD231
-	SOAD249
-	SOAD271
-	SOAD072
-	SOAD280
-	SOAD064
-	SOAD127
-	SOAD223
-	SOAD158
-	SOAD236
-	SOAD200
-	SOAD121
-	SOAD290
-	SOAD168
-	SOAD219
-	SOAD090
-	SOAD019
-	SOAD197
-	SOAD283
-	SOAD107
-	SOAD237
-	SOAD030
-	SOAD126
-	SOAD153
-	SOAD115
-	SOAD089
-	SOAD056
-	SOAD240
-	SOAD214
-	SOAD233
-	SOAD139
-	SOAD229
-	SOAD023
-	SOAD268
-	SOAD218
-	SOAD099
-	SOAD266
-	SOAD097
-	SOAD185
-	SOAD101

-	SOAD112
-	SOAD057
-	SOAD147
-	SOAD201
-	SOAD247
-	SOAD138
-	SOAD257
-	SOAD199
-	SOAD255
-	SOAD033
-	SOAD222
-	SOAD128
-	SOAD284
-	SOAD281
-	SOAD287
-	SOAD189
-	SOAD091
-	SOAD220
-	SOAD119
-	SOAD289
-	SOAD209
-	SOAD131
-	SOAD190
-	SOAD276
-	SOAD098
-	SOAD246
-	SOAD062
-	SOAD102
-	SOAD060
-	SOAD017
-	SOAD137
-	SOAD093
-	SOAD241
-	SOAD286
-	SOAD124
-	SOAD172
-	SOAD152
-	SOAD262
-	SOAD277
-	SOAD254
-	SOAD117
-	SOAD250

-	SOAD022
-	SOAD018
-	SOAD212
-	SOAD202
-	SOAD278
-	SOAD180
-	SOAD260
-	SOAD253
-	SOAD076
-	SOAD148
-	SOAD087
-	SOAD259
-	SOAD105
-	SOAD156
-	SOAD070
-	SOAD110
-	SOAD150
-	SOAD216
-	SOAD295
-	SOAD134
-	SOAD113
-	SOAD020
-	SOAD228
-	SOAD065
-	SOAD234
-	SOAD080
-	SOAD198
-	SOAD130
-	SOAD096
-	SOAD142
-	SOAD178
-	SOAD079
-	SOAD204
-	SOAD032
-	SOAD203
-	SOAD206
-	SOAD054
-	SOAD186
-	SOAD171
-	SOAD031
-	SOAD116
-	SOAD100

-	SOAD285
-	SOAD155
-	SOAD252
-	SOAD221
-	SOAD293
-	SOAD025
-	SOAD273
-	SOAD245
-	SOAD226
-	SOAD272
-	SOAD211
-	SOAD230
-	SOAD192
-	SOAD235
-	SOAD029
-	SOAD024
-	SOAD041
-	SOAD061
-	SOAD181
-	SOAD035
-	SOAD183
-	SOAD166
-	SOAD256
-	SOAD058
-	SOAD125
-	SOAD194
-	SOAD118
-	SOAD103
-	SOAD071
-	SOAD021
-	SOAD145
-	SOAD173
-	SOAD034
-	SOAD077
-	SOAD269
-	SOAD052
-	SOAD114
-	SOAD270
-	SOAD261
-	SOAD275
-	SOAD095
-	SOAD193

-	SOAD082
-	SOAD053
-	SOAD288
-	SOAD073
-	SOAD149
-	SOAD143
-	SOAD026
-	SOAD251
-	SOAD047
-	SOAD258
-	SOAD123
-	SOAD282
-	SOAD176
-	SOAD135
-	SOAD027
-	SOAD205
-	SOAD207
-	SOAD078
-	SOAD094
-	SOAD085
-	SOAD279
-	SOAD244
-	SOAD238
-	SOAD169
-	SOAD043
-	SOAD210
-	SOAD263
-	SOAD188
-	SOAD055
-	SOAD129
BSW00306	SOAD296
BSW00307	SOAD296
BSW00309	SOAD296
BSW00312	SOAD296
BSW00314	SOAD296
BSW00321	SOAD296
BSW00325	SOAD296
BSW00326	SOAD296
BSW00328	SOAD296
BSW00330	SOAD296
BSW00331	SOAD296
BSW00333	SOAD296

BSW00334	SOAD296
BSW00335	SOAD296
BSW00336	SOAD296
BSW00341	SOAD296
BSW00346	SOAD081, SOAD044
BSW00347	SOAD296
BSW00355	SOAD296
BSW00375	SOAD296
BSW00410	SOAD296
BSW00413	SOAD296
BSW00415	SOAD296
BSW00416	SOAD296
BSW00417	SOAD296
BSW00423	SOAD296
BSW00424	SOAD296
BSW00425	SOAD296
BSW00426	SOAD296
BSW00427	SOAD296
BSW00429	SOAD296
BSW00432	SOAD296
BSW00434	SOAD296
BSW005	SOAD296
BSW006	SOAD296
BSW010	SOAD296
BSW158	SOAD081, SOAD044
BSW160	SOAD296
BSW161	SOAD296
BSW162	SOAD296
BSW164	SOAD296
BSW168	SOAD296
BSW170	SOAD296
BSW172	SOAD296
BSW41900047	SOAD196

7 Functional specification

7.1 Overview

The TCP/IP concept of data transmission, particularly using Ethernet as the physical layer, has been established as a de-facto standard in the computing and telecommunication environments. The addressing of applications, logical addressing of end points and physical addressing are all covered in a layered suite of protocols and number assignments. Dynamic configuration and routing are at the core of the concepts implemented here.

AUTOSAR follows a concept of static communication relations pre-determined at compile time and rigid during run-time. The data transmitted is considered just as pre-determined as the source and sink that it needs to travel from and to.

The Socket Adaptor module aims at bridging the gap between these two concepts. By establishing a pre-determined table that includes the information required for AUTOSAR and leaving some items open to be updated by the TCP/IP stack during run-time the conflicting concepts are leveraged. Furthermore the SoAd decouples the call-back based software architecture from the socked based communication handling in the TCP/IP world.

In TCP/IP a particular ECU needs to be addressed using an IP address (and MAC address, if Ethernet is used). Furthermore multiple ECUs may be addressed by using broadcast or groupcast adresses. SoAd assumes that dynamic address allocation and mapping is done in the TCP/IP stack upon initialization. The Socket Connection Table is updated at this point to represent the current configuration and communication relations.

7.2 Use Cases

7.2.1 Autosar Signal Transmission

An AUTOSAR Signal will eventually take the shape of a PDU that needs to be transported over one of the attached communication systems (CAN, FlexRay, Lin, or TCP/IP). The decision on where to send a particular PDU is made in advance during system design and is executed by the PDU Router according to a fixed table during run-time. In all other busses used in AUTOSAR today (CAN, FlexRay, and Lin) the source does not need to address the sink directly, as all communication relations are known in advance and can be compressed into the PDU ID and a static routing table. The same concept is applied in the SoAd. In the connection tables described here each PDU ID is assigned to a socket connection. This needs to be done in two tables for each socket (one for the receive, the other for the transmit direction) as socket connections are always bi-directional.

7.2.2 Diagnostics over IP (DoIP)

DoIP is somewhat different from the AUTOSAR Signal Transmission use-case described above as these connections can not be fully pre-configured and are not active at all times of normal operation, but only during service. This requires a more dynamic approach to socket handling as described above. This is why the SoAd and DoIP share the same interfaces to the TCP/IP stack and other AUTOSAR modules, but handle their socket connections separately.
The protocols used for DoIP are envisioned to be specified in an ISO standards document and are not to be duplicated here.
The DoIP plug-in will be described in this document in more detail at a later time. For more information on the interfacing between the DoIP plug-in and SoAd, refer to chapter 5.6.

7.2.3 UDP Network Management (UdpNm)

The UDP Network Management uses the APIs described here to transmit network management messages used to control the power states of the ECUs connected to the same Ethernet communication domain. As this application requires a point-to-multipoint communication, it uses UDP as well as the broadcast mechanisms provided by IP and Ethernet.
UdpNm is described in a separate AUTOSAR document [17]. For more information on the interfacing between UdpNm and the SoAd, refer to chapter 5.5.

7.2.4 XCP on Ethernet

XCP is a protocol specified by ASAM which is used for calibration purpose. XCP makes use of APIs provided by the SoAd to exchange data between the ECU and the XCP master.

7.3 Concept of Operation

The SoAd will use connection oriented and connection less protocols of the TCP/IP stack to transfer data between ECUs. These protocols are well documented in the referenced literature and will not be re-explained in all detail here. However, some key considerations when using them in the AUTOSAR context are given in the following chapters.

7.3.1 The TCP/IP Protocol Family

The term TCP/IP is used to describe a family of communication protocols at various ISO/OSI model layers, including but not limited to: `IP`, `TCP`, `UDP`, `DHCP`, `ICMP`, `HTTP` and `ARP`.

7.3.2 TCP Connections

The Transmission Control Protocol (`TCP`) is defined in [24].

TCP is based on point-to-point communication relations. A broadcast or multicast is not possible in TCP.

TCP requires for one party to establish the connection and for the other to accept the incoming request. Two stations may establish multiple connections with each other, each will be handled by a different sockets and need to differ at least in one of the port numbers used.

If resource conservation for the SoAd is disabled, the initiator is pre-defined in the Socket Connection Table. Connection establishment is either done in the SoAd_SocketReset() function or handled by a resource management algorithm. Ideally, the connection is never lost until it is properly closed during SoAd_Shutdown. The SO_KEEPALIVE option may be used to this end.

[SOAD195] ⌈If a connection is lost during run-time or cannot be established during setup, a "*never give up*" strategy shall be implemented to continue connection attempts until shutdown. Unintentional loss of or failure to establish a connection shall be reported as a production error.⌋( )

[SOAD204] ⌈The "*never give up*" strategy shall not block the other functionality of the module, like e.g. shutdown.⌋( )

If resource conservation for the SoAd is enabled, each connection will be established when data is waiting to be transmitted and kept alive only if the resources are not required for a different socket.

In TCP all messages sent from a source to a sink are considered a continuous stream of consecutive bytes with preserved order. An acknowledgement scheme is in place to preserve the byte order spanning all messages. Messages are retransmitted by the source if the sink does not acknowledge reception within a certain time. TCP ensures data integrity (using checksums), byte order and completeness.
The PDU header option allows for transmission of multiple PDUs on one TCP connection.

7.3.2.1 Congestion Control

Due to the management overhead in TCP/IP packets it is deemed useful in most communications networks to pack a lot of payload into each packet to increase throughput. As this may result in a delay of data transmitted from a source, [27] describes an algorithm to transmit TCP packets depending available payload and time elapsed since the first data to transmit became available. This concept is commonly known as "*Nagles Algorithm*".

The nature of signals in AUTOSAR is such, that the accumulation of multiple values into a single message is not deemed useful.

[SOAD129] ⌈If a connection is intended to use the advantages of acknowledgement supplied by TCP, but does not whish to have its data delayed by the congestion controll process, it shall be able to deactivate this functionality of the TCP/IP stack. ⌋ ( )

7.3.3 UDP Communication

The User Datagram Protocol (UDP) is defined in [25].
For UDP there is no clear point-to-point communication relation required, multi- and broadcast of messages is possible. Due to this, no acknowledgement is sent by the sink(s), as the source might not know how many recipients there are.

There is no concept of consecutive messages or of a byte stream. Data integrity is ensured by using check sums.

The TCP/IP stack will transmit UDP packets as soon as possible, no congestion control is implemented.

UDP is used by AUTOSAR UDP NM [17] and others.

Although the PDU header option may be used in UDP as well, it is deemed far less useful there. Multiple PDUs may be received on a single listening socket even without the PDU header option, as source address and port may be used to de-multiplex the received PDUs.

7.3.4 Resource Management Option

In case each PDU is to be transmitted via a dedicated socket the TCP/IP stack will have to reserve buffer space for each of these sockets. This will lead to an enormous memory requirement and is deemed impractical for a larger number of PDUs.

[SOAD125] ⌈When a PDU is in line to be transmitted the resource management will check if this socket is properly set up to do so. If so, it will transmit the data there and increment this sockets counter in the Socket Connection Table. If the counter should overflow all counter values are integer divided by 2. ⌋ ( )

[SOAD126] ⌈Should the required socket not be available, the resource management will check for the lowest counter in the Socket Connection Table and try to close this socket, making sure no data waiting to be transmitted is lost in the process. It will then establish the socket connection required for the current PDU to be transmitted and initiate transmission. The newly created socket will (at least implicitly) use the resources of the one closed in the process. ⌋ ( )

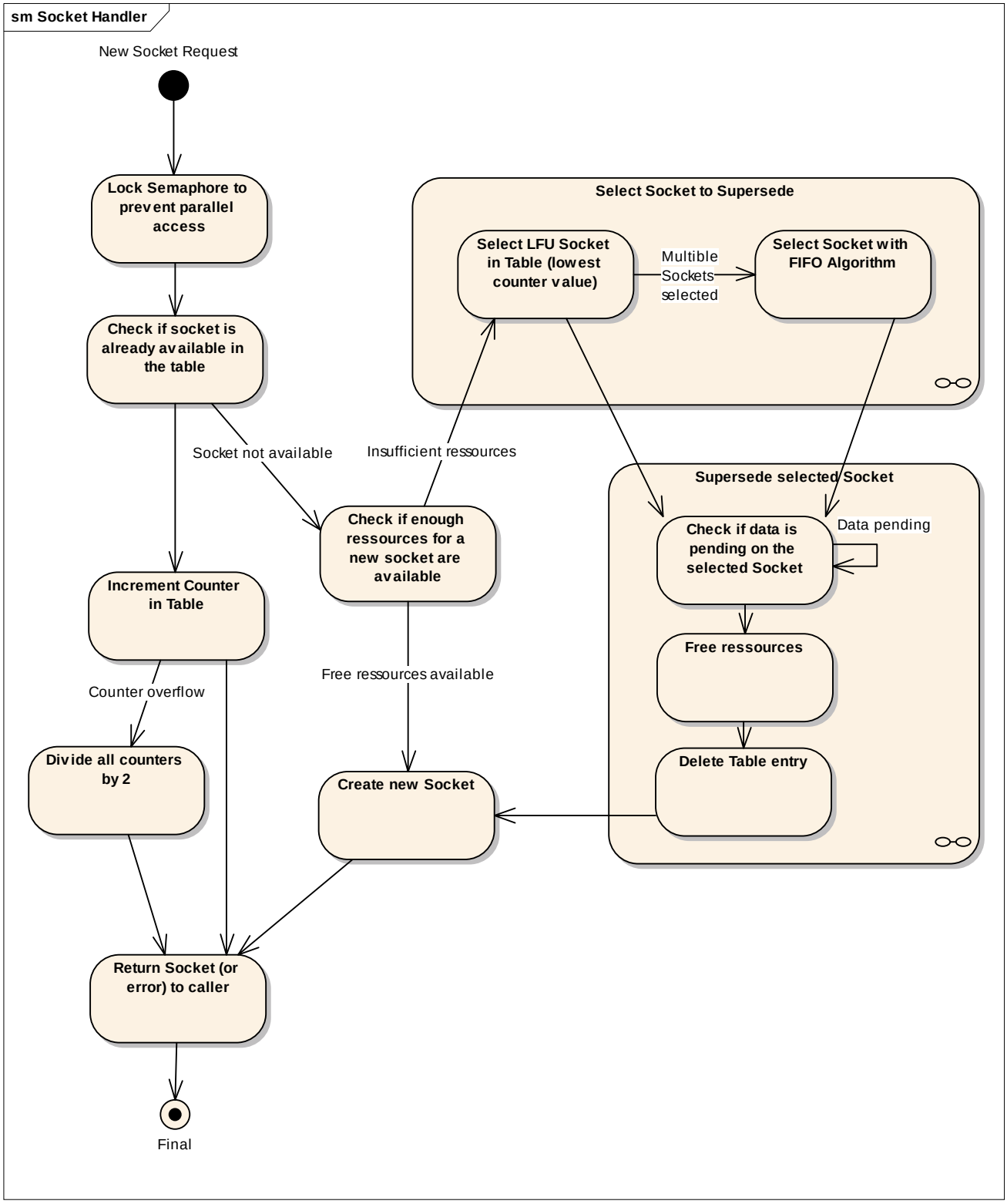


Figure 3: Socket Resource Management

7.3.5 PDU Header option

[SOAD197] ⌈If the PDU header option is enabled for a socket, a header shall be added ahead of the PDU data before transmission over the TCP/IP stack.⌋ ( )

[SOAD198] ⌈The header shall consist of a 4 byte ID field and a 4 byte length field. ⌋ ( )

[SOAD199] ⌈If the header option is enabled, the SDU length configuration in the socket connection table may be set to zero. In this case the comparison between the configured value in the socket connection table and the length given in the header shall be skipped. This is to allow for variable SDU length. ⌋ ( )

7.3.6 Byte-Order (Endianess)

The SoAd Module does not provide for adaption of the Bit- or Byte-Order (Endianess) within a PDU. System design is responsible for conversions in either the sending or the receiving application. The agreed byte order for IP connections needs to be observed in creating packet headers.

7.4 Services provided to upper layer

7.4.1 Initialization

[SOAD103] ⌈If SoAd is not initialized the SoAd module shall reject any call of a SoAd function with `E_NOT_OK`, except `SoAd_Init`. ⌋ ( )

[SOAD287] ⌈`SoAd_SocketReset()` shall ensure initialization of the TCP/IP stack. ⌋ ( )

[SOAD052] ⌈`SoAd_SocketReset()` shall initialize all TCP connections marked as initiator in the Socket connection table, if the resource conservation option is disabled. ⌋ ( )

[SOAD104] ⌈`SoAd_SocketReset()` shall set up listening ports for all UDP connections in the Socket connection table, if the resource conservation option is disabled. ⌋ ( )

[SOAD206] ⌈The implementation of `SoAd_SocketReset()` shall be done in such a way, that it will never block the overall initialization process. ⌋ ( )

[SOAD102] ⌈If AUTOSAR SoAd is not initialized the SoAd module shall not prohibit data traffic via the TCP/IP stack. ⌋ ( )

7.4.2 Shutdown

[SOAD053] ⌈`SoAd_Shutdown` shall try to close any open `TCP` connections and cancel any connection attempts.⌋( )

[SOAD054] ⌈`SoAd_Shutdown` shall try to ensure proper shutdown of the TCP/IP stack before returning.⌋( )

[SOAD207] ⌈The implementation of SOAD053 and SOAD054 shall be done in such a way, that `SoAd_Shutdown` will never block the overall shutdown process.⌋( )

7.5 Buffer handling

The TCP/IP stack is to handle its own memory resources and buffer management. The resources allocated need to be able to handle the maximum number of connections described in the Socket Connection Table (potentially modified by the resource management algorithm) plus any connections required by other modules.

[SOAD144] ⌈When using the AUTOSAR Call-back API to communicate with the TCP/IP stack, the SoAd shall not provide buffer for data storage or transfer. SoAd shall pass pointers to those buffers provided by the TCP/IP stack or the upper AUTOSAR layers.⌋( )

[SOAD184] ⌈When using the BSD socket API, the SoAd will have to provide buffers for the upper layer to retrieve the data from. The maximum available buffer size is defined by the configuration parameter `SOAD_BUFFER_MEMORY_SIZE`.⌋( )

7.6 Error Handling

7.6.1 Error classification

This section describes how the SoAd module has to manage the error classes that may occur during the life cycle of this basic software.

The general requirements document of AUTOSAR [3] specifies that all basic software modules must distinguish (according to the product life cycle) two error types:

- **Development errors:** these errors should be detected and fixed during the development phase. In most cases, these errors are software errors. The detection errors that should only occur during development can be switched off for production code (by static configuration, namely preprocessor switches).
- **Production errors:** these errors are hardware errors and software exceptions that cannot be avoided and are expected to occur in the production (i.e. series) code. This kind of error is commonly known as a run-time error.

[SOAD017] ⌈On errors and exceptions, the SoAd module shall not modify its current module state but shall simply report the error event to the DEM.⌋ ( )

[SOAD061] ⌈Values for production code Event Ids are assigned externally by the configuration of the DEM. They are published in the file `Dem_IntErrId.h` and included via `Dem.h`.⌋ ( )

[SOAD062] ⌈Development error values are of type `uint8`.⌋ ( )

[SOAD101] ⌈The following table lists development error IDs that can be detected within this software module which may also be derived from underlying protocol stacks:

Type or error	*Relevance*	*Related error code*	*Value*
API service called before initializing the module	Development	SOAD_E_NOTINIT	0x01
No such file or directory	Development	SOAD_E_NOENT	0x02
API service called with not allowed parameter value	Development	SOAD_WRONG_PARAM_VAL	0x03
API service called with NULL pointer	Development	SOAD_E_NULL_PTR	0x06
Bad file descriptor	Development	SOAD_E_BADF	0x09
Resource deadlock avoided	Development	SOAD_E_DEADLK	0x0B
Cannot allocate memory	Development	SOAD_E_NOMEM	0x0C
Permission denied	Development	SOAD_E_ACCES	0x0D
Not a directory	Development	SOAD_E_NOTDIR	0x14
Is a directory	Development	SOAD_E_ISDIR	0x15
Invalid argument	Development	SOAD_E_INVAL	0x16
Too many open files in system	Development	SOAD_E_NFILE	0x17
Too many open files	Development	SOAD_E_MFILE	0x18
Read-only file system	Development	SOAD_E_ROFS	0x1E
Numerical argument out of domain	Development	SOAD_E_DOM	0x21
Operation would block	Development	SOAD_E_WOULDBLOCK	0x22
Operation now in progress	Development	SOAD_E_INPROGRESS	0x24
Operation already in progress	Development	SOAD_E_ALREADY	0x25
Socket operation on non-socket	Development	SOAD_E_NOTSOCK	0x26
Destination address required	Development	SOAD_E_DESTADDRREQ	0x27
Message too long	Development	SOAD_E_MSGSIZE	0x28
Protocol wrong type for socket	Development	SOAD_E_PROTOTYPE	0x29
Protocol not available	Development	SOAD_E_NOPROTOOPT	0x2A
Protocol not supported	Development	SOAD_E_PROTONOSUPPORT	0x2B
Operation not supported	Development	SOAD_E_OPNOTSUPP	0x2D
Operation not supported	Development	SOAD_E_NOTSUP	0x2E
Address family not supported by protocol family	Development	SOAD_E_AFNOSUPPORT	0x2F
Address already in use	Development	SOAD_E_ADDRINUSE	0x30
Can't assign requested address	Development	SOAD_E_ADDRNOTAVAIL	0x31
No buffer space available	Development	SOAD_E_NOBUFS	0x37
Socket is already connected	Development	SOAD_E_ISCONN	0x38
Too many levels of symbolic links	Development	SOAD_E_LOOP	0x3D
File name too long	Development	SOAD_E_NAMETOOLONG	0x3F
No locks available	Development	SOAD_E_NOLCK	0x4D
Value too large to be stored in data type	Development	SOAD_E_OVERFLOW	0x54
Unknown error code from TCP/IP stack	Development	SOAD_E_TCPIPUNKNOWN	0x5A
PDU too long	Development	SOAD_E_PDU2LONG	0x5B
No AUTOSAR Connector	Development	SOAD_E_NOCONNECTOR	0x5C

configured			
Unknown PDU ID	Development	SOAD_E_INVALID_TXPDUID	0x5D
Invalid parameter pointer	Development	SOAD_E_PARAM_POINTER	0x5E

⌋( )

[SOAD232] ⌈The following table lists production error IDs that can be detected within this software module which may also be derived from underlying protocol stacks:

Type or error	*Relevance*	*Related error code*	*Value*
Interrupted system call	Production	SOAD_E_INTR	assigned by DEM
Input/output error	Production	SOAD_E_IO	assigned by DEM
Resource temporarily unavailable	Production	SOAD_E_AGAIN	assigned by DEM
Network is down	Production	SOAD_E_NETDOWN	assigned by DEM
Network is unreachable	Production	SOAD_E_NETUNREACH	assigned by DEM
Network dropped connection on reset	Production	SOAD_E_NETRESET	assigned by DEM
Software caused connection abort	Production	SOAD_E_CONNABORTED	assigned by DEM
Connection reset by peer	Production	SOAD_E_CONNRESET	assigned by DEM
Socket is not connected	Production	SOAD_E_NOTCONN	assigned by DEM
Operation timed out	Production	SOAD_E_TIMEDOUT	assigned by DEM
Connection refused	Production	SOAD_E_CONNREFUSED	assigned by DEM
Host is down	Production	SOAD_E_HOSTDOWN	assigned by DEM
No route to host	Production	SOAD_E_HOSTUNREACH	assigned by DEM
Broken pipe	Production	SOAD_E_PIPE	assigned by DEM
SDU length mismatch	Production	SOAD_E_SDULENGTH	assigned by DEM
No buffer available in upper layer	Production	SOAD_E_UPPERBUFF	assigned by DEM

⌋( )

7.6.2 Error detection

[SOAD056] ⌈The detection of development errors is configurable (ON / OFF) at pre-compile time. Setting the switch SOAD_DEV_ERROR_DETECT to TRUE shall activate or deactivate the detection of all development errors. ⌋( )

[SOAD057] ⌈If the SOAD_DEV_ERROR_DETECT switch is TRUE, the SoAd module shall enable the API parameter checking. The detailed description of the detected errors can be found in chapter 7.6.1 and chapter 7.7.2. ⌋( )

[SOAD058] ⌈The detection of production code errors cannot be switched off. ⌋( )

[SOAD183] ⌈Any failed call into the TCP/IP (BSD socket API) stack shall trigger an imediate call to `getlasterror()` and the result shall be reported to DET or DEM according to SOAD101 or SOAD232 respectively.

Note: As the enumerations of `SoAd_TcpIpErrorType` might not match the DEM error value, a mapping according to the related error code is required.

For further detail see section 8.3.4. ⌋ ( )

[SOAD185] ⌈When the AUTOSAR Call-Back API is implemented, the calls into the TCP/IP stack shall provide an error detection mechanism as defined in the SoAd main module. ⌋ ( )

7.6.3 Error notification

[SOAD130] ⌈Detected development errors shall be reported to the `Det_ReportError` service of the Development Error Tracer (DET) if the pre-processor switch `SOAD_DEV_ERROR_DETECT` is set. ⌋ ( )

[SOAD060] ⌈Production errors shall be reported to Diagnostic Event Manager [11]. ⌋ ( )

[SOAD202] ⌈If a fault occurs, the SoAd shall use the API `Dem_ReportErrorStatus` to inform the DEM module and shall set the parameter `EventStatus` to `FAILED`. ⌋ ( )

[SOAD203] ⌈If the reported fault disappears (not active/present), the SoAd shall use the API `Dem_ReportErrorStatus` to inform the DEM module and shall set the parameter `EventStatus` to `PASSED`. ⌋ ( )

[SOAD059] ⌈The SoAd module shall use the Development Error Tracer service [12]: `void Det_ReportError(ModuleId, InstanceId, ApiId, ErrorId)` to report development errors. ⌋ ( )

[SOAD186] ⌈When the AUTOSAR Call-Back API is implemented, the calls into the TCP/IP stack shall do error notification, just as the SoAd main module does. ⌋ ( )

7.7 Application notes

7.7.1 Wakeup notification

Wakeup notification is defined in detail in the ECU State Manager specification [9].

7.7.2 Debugging Concept

[SOAD077] ⌈Each variable that shall be accessible by AUTOSAR Debugging shall be defined as global variable.⌋ ( )

[SOAD078] ⌈All type definitions of variables which shall be debugged, shall be accessible by the header file `SoAd.h`.⌋ ( )

[SOAD079] ⌈The declaration of variables in the header file shall be such that it is possible to calculate the size of the variables by C-"sizeof".⌋ ( )

[SOAD080] ⌈Variables available for debugging shall be described in the respective Basic Software Module Description.⌋ ( )

7.8 Version checking

[SOAD295] ⌈The SoAd module shall perform Inter Module Checks to avoid integration of incompatible files.
The imported included files shall be checked by preprocessing directives.
The following version numbers shall be verified:
- <MODULENAME>_AR_RELEASE_MAJOR_VERSION
- <MODULENAME>_AR_RELEASE_MINOR_VERSION
Where <MODULENAME> is the module short name of the other (external) modules which provide header files included by the SoAd module.
If the values are not identical to the expected values, an error shall be reported.⌋ ( )

8 API specification

8.1 Imported types

The following types shall be imported by the SoAd from the modules given:

Module	*Imported Type*
Bsd	nfds_t
	size_t
	socklen_t
	ssize_t
ComM	ComM_ModeType
ComStack_Types	BufReq_ReturnType
	NetworkHandleType
	NotifResultType
	PduIdType
	PduInfoType
	PduLengthType
	RetryInfoType
Dem	Dem_EventIdType
	Dem_EventStatusType
GENERIC TYPES	...
Std_Types	Std_ReturnType
	Std_VersionInfoType

Module	*Imported Type*
ComStack_Types	PduIdType
	PduInfoType

The following types shall be imported by the BSD socket implementation from the modules given:

Module	*Imported Type*
GENERIC TYPES	...
SoAd	SoAd_DomainType
	SoAd_FcntlCmdType
	SoAd_FcntlFlagType
	SoAd_ProtocolType
	SoAd_RecvfromFlagType
	SoAd_SoOptionType
	SoAd_SocketType
	SoAd_TcpIpErrorType
	SoAd_SockAddrType
	SoAd_PollfdType

The following types shall be imported by the AUTOSAR call-back TCP/IP stack implementation from the modules given:

Module	*Imported Type*
ComStack_Types	BufReq_ReturnType
SoAd	SoAd_TcpIpPbufType
	SoAd_TcpIp_IpAddrPortType
Std_Types	Std_ReturnType
	Std_VersionInfoType

8.2 Type definitions

[SOAD065] ⌈`Enumeration` types shall match the `int` type of the TCP/IP stack implementation.⌋ ( )

8.2.1 SoAd_DomainType

[SOAD110] ⌈

Name:	`SoAd_DomainType`	
Type:	`Enumeration`	
Range:	`AF_INET`	0x02: Use IPv4
	`AF_INET6`	0x1c: Use IPv6
Description:	Address families allowed for SoAd.	

⌋ ( )

8.2.2 SoAd_ProtocolType

[SOAD111] ⌈

Name:	`SoAd_ProtocolType`	
Type:	`Enumeration`	
Range:	`IPPROTO_TCP`	0x06: Use TCP
	`IPPROTO_UDP`	0x11: Use UDP
	`SOL_SOCKET`	0xffff: options for socket level
Description:	Protocols to be used (TCP, UDP) in socket and level of configuration in setsockopt (TCP, SOL_SOCKET).	

⌋ ( )

8.2.3 SoAd_SocketType

[SOAD112] ⌈

Name:	`SoAd_SocketType`	
Type:	`Enumeration`	
Range:	`SOCK_STREAM`	0x01: Use streaming socket (TCP)
	`SOCK_DGRAM`	0x02: Use datagram socket (UDP)
Description:	Specifies the types of sockets to be used in SoAd.	

⌋ ( )

8.2.4 SoAd_SockAddrType

[SOAD113] ⌈

Name:	`SoAd_SockAddrType`		
Type:	`Structure`		
Element:	`uchar`	`sa_len`	Total length
	`SoAd_DomainType`	`sa_family`	Address family
	`char[16]`	`sa_data`	address value
Description:	Structure used to store most IP addresses.		

⌋ ( )

8.2.5 SoAd_PollFdType

[SOAD114] ⌈

Name:	SoAd_PollfdType		
Type:	Structure		
Element:	int	socket	Socket handle to be polled for events.
	SoAd_PollEventType	events	Events to be reported.
	SoAd_PollEventType	revents	Events found.
Description:	If poll() finds any of these requestable events set, they are copied to revents upon return.		

⌋ ( )

[SOAD030] ⌈The event request shall always set at least POLLERR, POLLHUP, and POLLNVAL. ⌋ ( )

8.2.6 SoAd_PollEventType

[SOAD115] ⌈

Name:	SoAd_PollEventType	
Type:	Enumeration	
Range:	POLLIN	0x0001: any readable data available
	POLLPRI	0x0002: OOB/Urgent readable data
	POLLOUT	0x0004: File descriptor is writeable
	POLLRDNORM	0x0040: non-OOB/URG data available
	POLLWRNORM = POLLOUT	0x0004: No write type differentiation
	POLLRDBAND	0x0080: OOB/Urgent readable data
	POLLWRBAND	0x0100: OOB/Urgent data can be written
	POLLERR	0x0008: some poll error occurred
	POLLHUP	0x0010: File descriptor was "hung up"
	POLLNVAL	0x0020: requested events "invalid"
Description:	Requestable events for poll function.	

⌋ ( )

8.2.7 SoAd_SoOptionType

[SOAD116] ⌈

Name:	SoAd_SoOptionType	
Type:	Enumeration	
Range:	SO_ACCEPTCONN	0x0002: socket has had listen()
	SO_ACCEPTFILTER	0x1000: there is an accept filter
	SO_BROADCAST	0x0020: permit sending of broadcast messages
	SO_DEBUG	0x0001: turn on debugging info recording
	SO_DONTROUTE	0x0010: just use interface addresses
	SO_KEEPALIVE	0x0008: keep connections alive
	SO_LINGER	0x0080: linger on close if data present
	SO_OOBINLINE	0x0100: leave received OOB data in line
	SO_RCVBUF	0x1002: receive buffer size
	SO_RCVLOWAT	0x1004: receive low-water mark
	SO_RCVTIMEO	0x1006: receive timeout
	SO_REUSEADDR	0x0004: allow local address reuse
	SO_REUSEPORT	0x0200: allow local address & port reuse
	SO_SNDBUF	0x1001: send buffer size

	SO_SNDLOWAT	0x1003: send low-water mark
	SO_SNDTIMEO	0x1005: send timeout
	SO_TIMESTAMP	0x0400: timesatmp received datagram traffic
	SO_USELOOPBACK	0x0040: bypass hardware when possible
	TCP_NODELAY	0x0001: don't delay send to coalesce packets
	TCP_MAXSEG	0x0002: set maximum segment size
	TCP_NOPUSH	0x0004: don't push last block of write
	TCP_NOOPT	0x0008: don't use TCP options
Description:	Options to set at Socket and TCP Level using the setsockopt function call.	

⌋( )

[SOAD055] ⌈Option SO_KEEPALIVE shall be set on socket level for all TCP connections marked PduR as destination in the Socket Connection Table.⌋( )

8.2.8 SoAd_TcpIpErrorType

[SOAD117] ⌈

Name:	SoAd_TcpIpErrorType	
Type:	Enumeration	
Range:	ENOENT	0x0002: No such file or directory
	EINTR	0x0004: Interrupted system call
	EIO	0x0005: Input/output error
	EBADF	0x0009: Bad file descriptor
	EDEADLK	0x000b: Resource deadlock avoided
	ENOMEM	0x000c: Cannot allocate memory
	EACCESS	0x000d: Permission denied
	ENOTDIR	0x0014: Not a directory
	EISDIR	0x0015: Is a directory
	EINVAL	0x0016: Invalid argument
	ENFILE	0x0017: Too many open files in system
	EMFILE	0x0018: Too many open files
	EROFS	0x001e: Read-only file system
	EPIPE	0x0020: Broken pipe
	EDOM	0x0021: Numerical argument out of domain
	EAGAIN	0x0023: Resource temporarily unavailable
	EWOULDBLOCK	0x0023: Operation would block
	EINPROGRESS	0x0024: Operation now in progress
	EALREADY	0x0025: Operation already in progress
	ENOTSOCK	0x0026: Socket operation on non-socket
	EDESTADDRREQ	0x0027: Destination address required
	EMSGSIZE	0x0028: Message too long
	EPROTOTYPE	0x0029: Protocol wrong type for socket
	ENOPROTOOPT	0x002a: Protocol not available
	EPROTONOSUPPORT	0x002b: Protocol not supported
	EOPNOTSUPP	0x002d: Operation not supported
	ENOTSUP	EOPNOTSUPP: Operation not supported
	EAFNOSUPPORT	0x002f: Address family not supported by protocol family
	EADDRINUSE	0x0030: Address already in use
	EADDRNOTAVAIL	0x0031: Can't assign requested address
	ENETDOWN	0x0032: Network is down
	ENETUNREACH	0x0033: Network is unreachable
	ENETRESET	0x0034: Network dropped connection on reset
	ECONNABORTED	0x0035: Software caused connection abort
	ECONNRESET	0x0036: Connection reset by peer
	ENOBUFS	0x0037: No buffer space available

	EISCONN	0x0038: Socket is already connected
	ENOTCONN	0x0039: Socket is not connected
	ETIMEDOUT	0x003c: Operation timed out
	ECONNREFUSED	0x003d: Connection refused
	ELOOP	0x003e: Too many levels of symbolic links
	ENAMETOOLONG	0x003f: File name too long
	EHOSTDOWN	0x0040: Host is down
	EHOSTUNREACH	0x0041: No route to host
	ENOLCK	0x004d: No locks available
	EOVERFLOW	0x0054: Value too large to be stored in data type
Description:	Error Codes returned by GetLastError().	

⌋ ( )

8.2.9 SoAd_FcntlFlagType

[SOAD118] ⌈

Name:	SoAd_FcntlFlagType	
Type:	Enumeration	
Range:	O_NONBLOCK	0x0004: no delay
Description:	Limits the flags argument to be used when calling fcntl.	

⌋ ( )

8.2.10 SoAd_FcntlCmdType

[SOAD119] ⌈

Name:	SoAd_FcntlCmdType	
Type:	Enumeration	
Range:	F_GETFL	0x0003: get file status flags
	F_SETFL	0x0004 set file status flags
Description:	Limits the cmd argument to be used when calling fcntl.	

⌋ ( )

8.2.11 SoAd_RecvfromFlagType

[SOAD142] ⌈

Name:	SoAd_RecvfromFlagType	
Type:	Enumeration	
Range:	MSG_OOB	0x01: process out-of-band data
	MSG_PEEK	0x02: peek at incoming message
	MSG_DONTROUTE	0x04: send without using routing tables
	MSG_EOR	0x08: data completes record
	MSG_TRUNC	0x10: data discarded before delivery
	MSG_CTRUNC	0x20: control data lost before delivery
	MSG_WAITALL	0x40: wait for full request or error
	MSG_DONTWAIT	0x80: this message should be non blocking
	MSG_EOF	0x100: data completes connection
Description:	Limits the flag argument to be used when calling recvfrom.	

⌋ ( )

8.2.12 SoAd_TcpIpEventType

[SOAD147] ⌈

Name:	SoAd_TcpIpEventType	
Type:	Enumeration	
Range:	RESET	0x01: TCP connection was reset
	CLOSED	0x02: TCP connection was closed successfully
	FIN_RECEIVED	0x03: A FIN signal was revceived on the TCP connection.
Description:	Describes the event reported by SoAd_TcpIpEvent.	

⌋ ( )

8.2.13 SoAd_TcpIpPbufType

[SOAD190] ⌈

Name:	SoAd_TcpIpPbufType		
Type:	Structure		
Element:	uint8*	payload	Pointer to the beginning of the payload data in pbuf. It may point to application data to be sent or to DMA area for received data.
	uint32	totLen	The total length in bytes of this pbuf + all following.
	uint16	len	The length of this pbuf in bytes.
Description:	Structure used to store TCP/IP packets.		

⌋ ( )

8.2.14 SoAd_TcpIp_IpAddrPortType

[SOAD192] ⌈

Name:	SoAd_TcpIp_IpAddrPortType		
Type:	Structure		
Element:	uint16	port	TCP/UDP port
	SoAd_SockAddrType	addr	Address bytes
Description:	Structure used to store IP address and port pairs.		

⌋ ( )

8.2.15 SoAd_ConfigType

[SOAD210] ⌈

Name:	SoAd_ConfigType		
Type:	Structure		
Element:	void	--	implementation specific.
Description:	This type shall contain the parameters of the container SoAd_GlobalConfig and its sub containers.		

⌋ ( )

8.3 Function definitions

This is a list of functions provided for upper layer modules.
Here is the API Naming convention for the SoAd services:
The service name format is `SoAd_<ServiceName>(…)`
`<ServiceName>`: is the name of the service primitive with first letter of each word upper case and consecutive letters lower case.
Service IDs used to identify the APIs within the SoAd module for DET and DEM error tracing are in the range from `0x00` to `0x7F`. The ID range from `0x80` to `0xFF` is reserved for the Autosar TCP/IP stacks internal usage, if the Autosar Call-back API is implemented. Not all IDs used by the Autosar TCP/IP stack are specified in detail in this document.
The TCP/IP stack uses the module ID of the SoAd when indicating DET or DET errors.
The BSD socket API calls do not report to DET or DEM, their IDs may be used to report errors.

8.3.1 General

8.3.1.1 SoAd_GetVersionInfo

[SOAD096] ⌈

Service name:	SoAd_GetVersionInfo	
Syntax:	`void SoAd_GetVersionInfo( Std_VersionInfoType* versioninfo )`	
Service ID[hex]:	0x0B	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	versioninfo	Pointer to where to store the version information of this module.
Return value:	None	
Description:	Returns the version information.	

This service returns the version information of this module. The version information includes:

- Module Id
- Vendor Id
- Vendor specific version numbers (BSW00407). ⌋ ( )

[SOAD171] ⌈This function shall be pre compile time configurable `On/Off` by the configuration parameter `SOAD_VERSION_INFO_API`. ⌋ ( )

[SOAD291] ⌈If development error detection is enabled and if the argument `versioninfo` is a NULL pointer, the function shall raise `SOAD_E_PARAM_POINTER` and return without any action. ⌋ ( )

8.3.1.2 DoIP_GetVersionInfo

[SOAD095] ⌈

Service name:	DoIP_GetVersionInfo	
Syntax:	`void DoIP_GetVersionInfo( Std_VersionInfoType* versioninfo )`	
Service ID[hex]:	0x60	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	versioninfo	Pointer to where to store the version information of this module.
Return value:	None	
Description:	Returns the version information.	

This service returns the version information of the DoIP plug-in. The version information includes:

- Module Id
- Vendor Id
- Vendor specific version numbers (BSW00407). ⌋ ( )

[SOAD172] ⌈This function shall be pre compile time configurable `On/Off` by the configuration parameter: `DOIP_VERSION_INFO_API`. ⌋ ( )

[SOAD292] ⌈If development error detection is enabled and if the argument `versioninfo` is a NULL pointer, the function shall raise `SOAD_E_PARAM_POINTER` and return without any action. ⌋ ( )

8.3.1.3 TcpIp_GetVersionInfo

[SOAD094] ⌈

Service name:	TcpIp_GetVersionInfo	
Syntax:	`void TcpIp_GetVersionInfo( Std_VersionInfoType* versioninfo )`	
Service ID[hex]:	0x8A	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	versioninfo	Pointer to where to store the version information of this module.
Return value:	None	
Description:	Returns the version information.	

⌋ ( )

[SOAD145] ⌈This service is only available if the AUTOSAR call-back API is available.
In this case the TCP/IP stack shall

This service returns the version information of the TCP/IP stack. The version information includes:

- Module Id
- Vendor Id
- Vendor specific version numbers (BSW00407). ⌋( )

[SOAD173] ⌈This function shall be pre compile time configurable `On/Off` by the configuration parameter: `TCPIP_VERSION_INFO_API`⌋( )

[SOAD148] ⌈This requires for the TCP/IP stack to publicize the information requested above. ⌋( )

[SOAD293] ⌈If development error detection is enabled and if the argument `versioninfo` is a NULL pointer, the function shall raise `SOAD_E_PARAM_POINTER` and return without any action. ⌋( )

8.3.1.4 SoAd_ChangeParameter

[SOAD189] ⌈The PduR shall not use the SoAd API to change an STMIN value, as such a value is not used in SoAd. ⌋( )

8.3.1.5 TcpIp_SetDhcpHostNameOption

[SOAD196] ⌈

Service name:	TcpIp_SetDhcpHostNameOption	
Syntax:	`Std_ReturnType TcpIp_SetDhcpHostNameOption( uint8* HostNameOption, uint8 HostNameLen )`	
Service ID[hex]:	0x89	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	HostNameOption	DHCP Host Name Option according to ISO 13400.
	HostNameLen	Length of the data to be set.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The request has been accepted E_NOT_OK: The request has not been accepted.
Description:	This API sets the DHCP Host Name Option according to ISO 13400. The DHCP Host Name Option may consist of static and dynamic content. The static content will usually be found in SOAD053_Conf: SoAdDoIpHostNameOpt. This API needs to be implemented whenever DoIP is to be supported, independent of the API used.	

⌋(BSW41900047)

[SOAD218] ⌈
If development error detection is enabled: the function shall check that the service `TcpIp_Init` was previously called. If the check fails, the function shall raise the development error `SOAD_E_NOTINIT` and return `E_NOT_OK`. ⌋( )

[SOAD228] ⌈
If development error detection is enabled: the function shall check parameter `HostNameOption` for being a `NULL_PTR`. If this is `TRUE`, the function shall raise the development error `SOAD_E_NULL_PTR` and return `E_NOT_OK`. ⌋ ( )

[SOAD227] ⌈
If development error detection is enabled: the function shall check parameter `HostNameOption` for being valid. If the check fails, the function shall raise the development error `SOAD_E_PARAM_POINTER` and return `E_NOT_OK`. ⌋ ( )

[SOAD229] ⌈
If development error detection is enabled: the function shall check parameter `HostNameLen` for being valid. If the check fails, the function shall raise the development error `SOAD_E_INVAL` and return `E_NOT_OK`. ⌋ ( )

[SOAD226] ⌈
Caveat: The function requires previous initialization (`TcpIp_Init`). ⌋ ( )

8.3.2 Initialization and Shutdown

8.3.2.1 Initialization and Shutdown of the TCP/IP Stack

As Initialization and Shutdown of the TCP/IP Stack is not specified in the IEEE 1003.1 document and since the TCP/IP Stack is not to be an AUTOSAR SW module, there will be function calls specific to a certain TCP/IP Stack implementation. These shall be called at the appropriate time in the SoAds Initialization and Shutdown functions.

8.3.2.2 TcpIp_Init

[SOAD193] ⌈

Service name:	TcpIp_Init	

Syntax:	`void TcpIp_Init(`
	`  void`
	`)`	

Service ID[hex]:	0x80	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Return value:	void	None
Description:	This service initializes the TCP/IP Stack. TcpIp_Init may not block the start-up process for an indefinite amount of time. Caveats: The call of this service is mandatory before using the TcpIp instance for further processing.	

⌋( )

[SOAD230] ⌈

Caveat: The API has to be called during initialization. ⌋( )

8.3.2.3 TcpIp_Shutdown

[SOAD194] ⌈

Service name:	TcpIp_Shutdown	

Syntax:	`void TcpIp_Shutdown(`
	`  void`
	`)`	

Service ID[hex]:	0x81	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Return value:	void	None
Description:	This service closes all pending transport protocol connections, releases all resources and stops the TCP/IP stack.	

⌋( )

8.3.2.4 SoAd_Init

[SOAD093] ⌈

Service name:	SoAd_Init	

Syntax:	`void SoAd_Init(`
	`  const SoAd_ConfigType* SoAdConfigPtr`
	`)`	

Service ID[hex]:	0x01	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	SoAdConfigPtr	Points to the implementation specific structure.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Description: This service initializes all global variables of a Socket Adaptor instance and puts it into the idle state. It has no return value because software errors in initialization data shall be detected during configuration time (e.g. by configuration tool). Furthermore, if a hardware error occurs it shall be reported via the error manager modules. Caveats: The call of this service is mandatory before using the SoAd instance for further processing. The API has to be called during initialization.	

⌋( )

[SOAD211] ⌈
`SoAd_Init` shall store the access to the configuration structure for subsequent API calls. ⌋( )

[SOAD212] ⌈
Configuration: The user shall pass the post-build configuration or a `NULL_PTR` as parameter depending on the configuration variant. ⌋( )

[SOAD215] ⌈
If development error detection is enabled: the function shall check the parameter `SoAdConfigPtr` for being valid. If the check fails, the function shall raise the development error `SOAD_E_PARAM_POINTER`. ⌋( )

[SOAD216] ⌈
If development error detection is enabled: the function shall check the parameter `SoAdConfigPtr` for containing a valid configuration. If the check fails, the function shall raise the development error `SOAD_E_INVAL`. ⌋( )

8.3.2.5 SoAd_Shutdown

[SOAD092] ⌈

Service name:	SoAd_Shutdown	
Syntax:	`Std_ReturnType SoAd_Shutdown( void )`	
Service ID[hex]:	0x09	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The request has been accepted and executed fully and correctly. E_NOT_OK: The request has not been executed correctly. SOAD_E_INPROGRESS: The request has not been executed fully, but no error has occured so far.
Description:	This service atempts to close all pending transport protocol connections, frees all resources and stops the SoAd Module. It will not block the overall shutdown process, but return SOAD_E_INPROGRESS, if the request could not be fully executed, while no error has occured.	

⌋ ( )

8.3.2.6 SoAd_SocketReset

[SOAD127] ⌈

Service name:	SoAd_SocketReset
Syntax:	`void SoAd_SocketReset( void )`
Service ID[hex]:	0x07
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This service shall initiate setup of all TCP connections which are labeled for this ECU to be the initiator. All other UDP and TCP Sockets will be put to the listen state. If called after initialization this service shall close all open connections and terminate ongoing connection setups. As well as close all open listening ports. If the Rescource Conservation Option is used, all counters shall be reset. This function shall not block, if socket or connection setup/closure fails or is delayed. It has no return value because software errors in initialization data shall be detected during configuration time (e.g. by configuration tool). Furthermore, if a hardware error occurs it shall be reported via the error manager modules.

⌋ ( )

[SOAD220] ⌈
If development error detection is enabled: the function shall check that the service `SoAd_Init` was previously called. If the check fails, the function shall raise the development error `SOAD_E_NOTINIT`. ⌋ ( )

[SOAD231] ⌈
If development error detection is enabled: the function shall raise development errors according to SOAD101. ⌋ ( )

[SOAD233] ⌈
The function shall raise production errors according to SOAD232. ⌋ ( )

[SOAD222] ⌈
Caveat: The function requires previous initialization (`SoAd_Init`). ⌋ ( )

8.3.3 Normal Operation

8.3.3.1 SoAdIf_Transmit

[SOAD091] ⌈

Service name:	SoAdIf_Transmit	
Syntax:	`Std_ReturnType SoAdIf_Transmit( PduIdType SoAdSrcPduId, const PduInfoType* SoAdSrcPduInfoPtr )`	
Service ID[hex]:	0x08	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	SoAdSrcPduId	This parameter contains a unique identifier referencing to the PDU Routing Table and thereby specifiying the socket to be used for tranmission of the data.
	SoAdSrcPduInfoPtr	A pointer to a structure with socket related data: data length and pointer to a data buffer.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The request has been accepted E_NOT_OK: The request has not been accepted, e.g. due to a still ongoing transmission in the corresponding socket or the to be transmitted message is too long.
Description:	This service initiates a request for transmission of the L-PDU specified by the SoAdSrcPduId. The corresponding socket has to be resolved by the SoAdSrcPduId. This call is used to mimic the call to an IF in AUTOSAR. Development errors: Invalid values of SoAdSrcPduId or SoAdSrcPduInfoPtr will be reported to the development error tracer (SOAD_E_INVALID_TXPDUID or SOAD_E_PARAM_POINTER).	

⌋ ( )

[SOAD213] ⌈

If development error detection is enabled: the function shall check that the service SoAd_Init was previously called. If the check fails, the function shall raise the development error SOAD_E_NOTINIT and return E_NOT_OK. ⌋( )

[SOAD234] ⌈
If development error detection is enabled: the function shall check parameter SoAdSrcPduInfoPtr for being a NULL_PTR. If this is TRUE, the function shall raise the development error SOAD_E_NULL_PTR and return E_NOT_OK. ⌋( )

[SOAD235] ⌈
If development error detection is enabled: the function shall check parameter SoAdSrcPduInfoPtr for being valid. If the check fails, the function shall raise the development error SOAD_E_PARAM_POINTER and return E_NOT_OK. ⌋( )

[SOAD214] ⌈
If development error detection is enabled: the function shall check parameter SoAdSrcPduId for being valid. If the check fails, the function shall raise the development error SOAD_E_INVAL and return E_NOT_OK. ⌋( )

[SOAD217] ⌈
Caveat: The function requires previous initialization (SoAd_Init). ⌋( )

8.3.3.2 SoAdTp_Transmit

[SOAD105] ⌈

Service name:	SoAdTp_Transmit	

Syntax:	`Std_ReturnType SoAdTp_Transmit(`
	`  PduIdType SoAdSrcPduId,`
	`  const PduInfoType* SoAdSrcPduInfoPtr`
	`)`	

Service ID[hex]:	0x0F	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	SoAdSrcPduId	This parameter contains a unique identifier referencing to the PDU Routing Table and thereby specifiying the socket to be used for tranmission of the data.
	SoAdSrcPduInfoPtr	A pointer to a structure with socket related data. Only the length data is valid.

Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The request has been accepted E_NOT_OK: The request has not been accepted, e.g. due to a still ongoing transmission in the corresponding socket or the to be transmitted message is too long.
Description:	This service is utilized to request the transfer of data. It sets a flag for indicating that a transmit request is present. This function has to be called with the PDU-ID of the SoAd, i.e. the upper layer has to translate its own PDU-ID into the one of the SoAd for the corresponding message. This call shall fail if no entry for the PDU-ID can be found in the socket connection table. Within the provided SoAdSrcPduInfoPtr only SduLength is valid (no data)! If this function returns E_OK then there will arise a call of PduR_SoAdCopyTxData in order to get data for sending. This call is used to mimic the call to a TP in AUTOSAR.	

⌋ ( )

[SOAD224] ⌈
If development error detection is enabled: the function shall check that the service `SoAd_Init` was previously called. If the check fails, the function shall raise the development error `SOAD_E_NOTINIT` and return `E_NOT_OK`. ⌋ ( )

[SOAD225] ⌈
If development error detection is enabled: the function shall check parameter `SoAdSrcPduInfoPtr` for being a `NULL_PTR`. If this is `TRUE`, the function shall raise the development error `SOAD_E_NULL_PTR` and return `E_NOT_OK`. ⌋ ( )

[SOAD236] ⌈
If development error detection is enabled: the function shall check parameter `SoAdSrcPduInfoPtr` for being valid. If the check fails, the function shall raise the development error `SOAD_E_PARAM_POINTER` and return `E_NOT_OK`. ⌋ ( )

[SOAD237] ⌈
If development error detection is enabled: the function shall check parameter `SoAdSrcPduId` for being valid. If the check fails, the function shall raise the development error `SOAD_E_INVAL` and return `E_NOT_OK`. ⌋ ( )

[SOAD223] ⌈
Caveat: The function requires previous initialization (`SoAd_Init`). ⌋ ( )

8.3.4 BSD Socket API (COTS) functions used by the SoAd

[SOAD025] ⌈The implementation of the BSD Socket API is to use IEEE Std 1003.1 [21] as a prototype. This chapter lists those functions required to be implemented by the SoAd and gives additional information (e.g. types) or limitations required for inclusion in AUTOSAR. This document shall supersede the IEEE document in case of contradictions only. Where additional information is given in [21] and not expicitly over-ruled here, it shall still be binding for the implementation.⌋ ( )

[SOAD136] ⌈All calls to the BSD Socket interface shall be executed non-blocking [`O_NONBLOCK`].⌋ ( )

[SOAD239] ⌈As the BSD-TCP/IP-Stack is not an AUTOSAR Module, the calling module shall catch errors (usually indicated by `Return value == -1`) and call `getlasterror()` to determine the error code.⌋ ( )

[SOAD238] ⌈The calling module shall report production errors according to SOAD232 to DEM.⌋ ( )

The note following SOAD183 applies.

[SOAD240] ⌈If development error detection is enabled, the calling function shall report development errors according to SOAD101 to DET.⌋ ( )

8.3.4.1 accept

[SOAD034] ⌈

Service name:	accept	

Syntax:	`int accept(`
	`  int socket,`
	`  struct SoAd_SockAddrType* restrict address,`
	`  socklen_t* restrict address_len`
	`)`	

Service ID[hex]:	0x22	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	socket	Specifies a socket that was created with socket, has been bound to an address with bind, and has issued a successful call to listen.
Parameters (inout):	address_len	Points to a socklen_t structure which on input specifies the length of the supplied SoAd_sockaddrType structure, and on output specifies the length of the stored address.
Parameters (out):	address	A pointer to a SoAd_sockaddrType structure where the address of the connecting socket shall be returned.
Return value:	int	Upon successful completion, accept shall return the non-negative file descriptor [0..*] of the accepted socket. Otherwise, -1 shall be returned.
Description:	The accept function shall extract the first connection on the queue of pending connections, create a new socket with the same socket type protocol and address family as the specified socket, and allocate a new file descriptor for that socket.	

⌋ ( )

[SOAD035] ⌈A connection shall only be accepted if there is an appropriate entry in the Socket Connection Table that does not have a socket assigned to it already.⌋ ( )

[SOAD036] ⌈If a connection is not marked with destination PduR, UdpNm or Xcp in the Socket Connection Table it shall be accepted by the destination listed there, not by the SoAd itself.⌋ ( )

[SOAD039] ⌈After accepting a connection the Socket Connection Table shall be updated with the Socket Handle and any other information required.⌋ ( )

8.3.4.2 bind

[SOAD027] ⌈

Service name:	bind	

Syntax:	`int bind(`
	`  int socket,`
	`  const struct SoAd_SockAddrType* address,`
	`  socklen_t address_len`
	`)`	

Service ID[hex]:	0x1B	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	socket	Specifies the file descriptor of the socket to be bound.
	address	Points to a SoAd_SockAddrType structure containing the address to be bound to the socket. The length and format of the address depend on the address family of the socket.
	address_len	Specifies the length of the SoAd_SockAddrType structure pointed to by the address argument.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	int	Upon successful completion, bind shall return 0; otherwise, -1 shall be returned.
Description:	The bind function shall assign the local socket address specified in address to a socket identified by descriptor socket that has no local socket address assigned. Sockets created with the socket function are initially unnamed; they are identified only by their address family.	

⌋ ( )

8.3.4.3 close

[SOAD047] ⌈

Service name:	close	

Syntax:	`int close(`
	`  int socket`
	`)`	

Service ID[hex]:	0x19	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	socket	Specifies the file descriptor associated with the socket to be closed.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	int	Upon successful completion, close shall return 0; otherwise, -1 shall be returned.
Description:	The close function shall de-allocate the file descriptor indicated by socket.	

⌋ ( )

8.3.4.4 connect

[SOAD021] ⌈

Service name:	connect	

Syntax:	`int connect(`
	`  int socket,`
	`  const struct SoAd_SockAddrType* address,`
	`  socklen_t address_len`
	`)`	

Service ID[hex]:	0x15	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	socket	Specifies the file descriptor associated with the socket.
	address	Points to a SoAd_sockaddrType structure containing the peer address. The length and format of the address depend on the address family of the socket.
	address_len	Specifies the length of the SoAd_sockaddrType structure pointed to by the address argument.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	int	Upon successful completion, connect shall return 0; otherwise, -1 shall be returned. As O_NONBLOCK is the normal mode of operation in the SoAd, see SOAD123.
Description:	If the socket has not already been bound to a local address, connect shall bind it to an address which is an unused local address. If the initiating socket is not connection-mode, then connect shall set the sockets peer address, and no connection is made. For SOCK_DGRAM sockets, the peer address identifies where all datagrams are sent on subsequent send functions. If address is a null address for the protocol, the sockets peer address shall be reset. If the connection cannot be established immediately and O_NONBLOCK is set for the file descriptor for the socket, connect() shall fail [EINPROGRESS], but the connection request shall not be aborted, and the connection shall be established asynchronously. Subsequent calls to connect() for the same socket, before the connection is established, shall fail [EALREADY]. When the connection has been established asynchronously, select() and poll() shall indicate that the file descriptor for the socket is ready for writing. Note: For non-blocking calls [O_NONBLOCK], a failure is the normal response.	

⌋ ( )

8.3.4.5 fcntl

[SOAD031] ⌈

Service name:	`Fcntl`

Synopsis:	`int fcntl(`
	`  int socket,`
	`  int cmd,`
	`  ...)`

Service ID [hex]:	0x1F
Sync/Async:	Synchronous
Reentrancy:	Reentrant

⌋ ( )

[SOAD064] ⌈As this function call uses a variable number of arguments, which is not permissible in MISRA C, the call shall be restricted to the following syntax, but the synopsis needs to conform to a call with multiple arguments (see above). ⌋ ( )

[SOAD205] ⌈

Service name:	fcntl	

Syntax:	`int fcntl(`
	`  int socket,`
	`  SoAd_FcntlCmdType cmd,`
	`  SoAd_FcntlFlagType flags`
	`)`	

Service ID[hex]:	0x1F	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	socket	Specifies the file descriptor associated with the socket.
	cmd	Codifies a command to the socket.
	flags	Flags to be passed with the command issued.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	int	Upon successful completion, the value returned shall depend on cmd as follows: F_GETFL Value of file status flags and access modes. The return value is not negative [0...0x####]. F_SETFL Value other than -1.
Description:	Send commands to the TCP/IP stack for configuration. Or read status of configuration flags.	

⌋ ( )

8.3.4.6 getlasterror

[SOAD042] ⌈

Service name:	getlasterror	
Syntax:	`SoAd_TcpIpErrorType getlasterror( void )`	
Service ID[hex]:	0x20	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Return value:	SoAd_TcpIpErrorType	The return value indicates the error code for the last Socket API routine performed.
Description:	This function returns the last network error that occurred.	

⌋( )

[SOAD041] ⌈If the BSD approach is used, the SoAd module shall implement the API `GetLastError()` to receive the last network error from the TCP/IP stack. ⌋( )
Note: With the AUTOSAR it is not possible to access error codes via the global `errno` variable as in the BSD approach.

8.3.4.7 listen

[SOAD024] ⌈

Service name:	listen	
Syntax:	`int listen( int socket, int backlog )`	
Service ID[hex]:	0x18	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	socket	Specifies the file descriptor associated with the socket.
	backlog	The backlog argument provides a hint to the implementation which the implementation shall use to limit the number of outstanding connections in the socket's listen queue. Implementations may impose a limit on backlog and silently reduce the specified value. Normally, a larger backlog argument value shall result in a larger or equal length of the listen queue.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	int	Upon successful completion, listen shall return 0; otherwise, -1 shall be returned.
Description:	The listen function shall mark a connection-mode socket, specified by the socket argument, as accepting connections.	

⌋( )

8.3.4.8 poll

[SOAD029] ⌈

Service name:	poll	

Syntax:	`int poll(`
	`  struct SoAd_PollfdType fds[],`
	`  nfds_t nfds,`
	`  int timeout`
	`)`	

Service ID[hex]:	0x1D	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	nfds	The number of SoAd_pollfdType structures in the fds array is specified by nfds.
	timeout	If none of the defined events have occurred on any selected file descriptor, poll() shall wait at least timeout milliseconds for an event to occur on any of the selected file descriptors. If the value of timeout is 0, shall return immediately. If the value of timeout is -1, poll() shall block until a requested event occurs or until the call is interrupted.
Parameters (inout):	fds[]	The fds argument specifies the file descriptors to be examined and the events of interest for each file descriptor. It is a pointer to an array with one member for each open file descriptor of interest. The array's members are SoAd_pollfdType structures.
Parameters (out):	None	
Return value:	int	Upon successful completion, poll shall return a non-negative value. A positive value indicates the total number of file descriptors that have been selected (that is, file descriptors for which the revents member is non-zero). A value of 0 indicates that the call timed out and no file descriptors have been selected. Upon failure, poll shall return -1.
Description:	The poll() function provides SoAd with a mechanism for multiplexing input/output over a set of file descriptors. For each member of the array pointed to by fds, poll() shall examine the given file descriptor for the event(s) specified in events. The number of SoAd_pollfdType structures in the fds array is specified by nfds. The poll() function shall identify those file descriptors on which an application can read or write data, or on which certain events have occurred.	

⌋ ( )

8.3.4.9 recvfrom

[SOAD023] ⌈

Service name:	recvfrom	

Syntax:	`ssize_t recvfrom(`
	`  int socket,`
	`  void* restrict buffer,`
	`  size_t length,`
	`  SoAd_RecvfromFlagType flags,`
	`  struct SoAd_SockAddrType* restrict address,`
	`  socklen_t* restrict address_len`
	`)`	

Service ID[hex]:	0x17	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	socket	Specifies the file descriptor associated with the socket.
	buffer	Points to the buffer where the message should be stored.
	length	Specifies the length in bytes of the buffer pointed to by the buffer argument.
	flags	Specifies the type of message reception. Values of this argument are formed by logically OR'ing zero or more values.
	address	Points to a SoAd_sockaddrType structure in which the sending address is to be stored. The length and format of the address depend

		on the address family of the socket.
	address_len	Specifies the length of the SoAd_sockaddrType structure pointed to by the address argument.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	ssize_t	Upon successful completion, recvfrom shall return the length of the message in bytes [0x0000 ... 0x####]. If no messages are available to be received and the peer has performed an orderly shutdown, recvfrom shall return 0. Otherwise, the function shall return -1 to indicate an error.
Description:	The recvfrom function shall receive a message from a connection-mode or connectionless-mode socket. Caveats: For message-based sockets, such as SOCK_DGRAM, the entire message shall be read in a single operation. If a message is too long to fit in the supplied buffer, and MSG_PEEK is not set in the flags argument, the excess bytes shall be discarded. For stream-based sockets, such as SOCK_STREAM, message boundaries shall be ignored, and no data shall be discarded.	

⌋( )

[SOAD028] ⌈As `recvfrom` will not filter for the source address of a packet, SoAd shall discard all packets that do not match an entry in the socket connection table.⌋( )

8.3.4.10 sendto

[SOAD022] ⌈

Service name:	sendto	

Syntax:	`ssize_t sendto(`
	`  int socket,`
	`  const void* message,`
	`  size_t length,`
	`  int flags,`
	`  const struct SoAd_SockAddrType* dest_address,`
	`  socklen_t dest_len`
	`)`	

Service ID[hex]:	0x16	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	socket	Specifies the file descriptor associated with the socket.
	message	Points to a buffer containing the message to be sent.
	length	Specifies the size of the message in bytes.
	flags	Specifies the type of message transmission.
	dest_address	Points to a SoAd_SockAddrType structure containing the destination address. The length and format of the address depend on the address family of the socket.
	dest_len	Specifies the length of the SoAd_sockaddrType structure pointed to by the dest_addr argument.

Parameters (inout):	None	
Parameters (out):	None	
Return value:	ssize_t	Upon successful completion, sendto() shall return the number of bytes sent. Otherwise, -1 shall be returned.
Description:	The sendto() function shall send a message through a connection-mode or connectionless-mode socket. If the socket is connectionless-mode, the message shall be sent to the address specified by dest_addr. If the socket is connection-mode, dest_addr shall be ignored.	

⌋ ( )

8.3.4.11 setsockopt

[SOAD033] ⌈

Service name:	setsockopt	

Syntax:	`int setsockopt(`
	`  int socket,`
	`  SoAd_ProtocolType level,`
	`  SoAd_SoOptionType option_name,`
	`  const void* option_value,`
	`  socklen_t option_len`
	`)`	

Service ID[hex]:	0x21	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	socket	Specifies the file descriptor associated with the socket.
	level	The level argument specifies the protocol level at which the option resides.
	option_name	Specifies the option to be set.
	option_value	Value the option is to be set to.
	option_len	Numer of bytes occupied by the value for this option.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	int	Upon successful completion, setsockopt shall return 0. Otherwise, -1 shall be returned.
Description:	The setsockopt function shall set the option specified by the option_name argument, at the protocol level specified by the level argument, to the value pointed to by the option_value argument for the socket associated with the file descriptor specified by the socket argument.	

⌋ ( )

8.3.4.12 socket

[SOAD026] ⌈

Service name:	socket	

Syntax:	`int socket(`
	`  SoAd_DomainType domain,`
	`  SoAd_SocketType type,`
	`  SoAd_ProtocolType protocol`
	`)`	

Service ID[hex]:	0x1A	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	domain	Specifies the address family in which a socket is to be created.
	type	Specifies the type of socket to be created.
	protocol	Specifies a particular protocol to be used with the socket. Specifying a protocol of 0 in order to use an unspecified default is not permitted and shall result in a development error.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	int	Upon successful completion, socket shall return a non-negative integer, the socket file descriptor. Otherwise, a value of -1 shall be returned.
Description:	The socket function shall create an unbound socket in a communications domain, and return a file descriptor (called socket) that can be used in later function calls that operate on sockets.	

⌋ ( )

8.3.5 AUTOSAR socket API functions used by the SoAd

The following services of the TCP/IP Stack are called by the SoAd if the AUTOSAR socket API is used.

8.3.5.1 TcpIp_ProvideTxBuffer

[SOAD187] ⌈

Service name:	TcpIp_ProvideTxBuffer	

Syntax:	`BufReq_ReturnType TcpIp_ProvideTxBuffer(`
	`  int SoHandle,`
	`  SoAd_TcpIp_IpAddrPortType Destination,`
	`  SoAd_TcpIpPbufType** PbufPtr,`
	`  uint32 Length`
	`)`	

Service ID[hex]:	0x88	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant	
Parameters (in):	SoHandle	This parameter contains the socket handle on which the data is to be transmitted.
	Destination	IP address and port where data is to be sent to
	Length	Size of the Buffer to be provided for data

Parameters (inout):	None	
Parameters (out):	PbufPtr	Buffer provided for data to be transmitted.
Return value:	BufReq_ReturnType	E_OK: The request has been accepted E_NOT_OK: The request has not been accepted, e.g. due to a still ongoing transmission in the corresponding socket or the to be transmitted message is too long or the socket is not connected (for TCP).
Description:	This service is utilized to request the transfer of data. The Buffer to which Data points must not be modified until the transmission confirmation was received.	

⌋ ( )

[SOAD219] ⌈
If development error detection is enabled: the function shall check that the service `TcpIp_Init` was previously called. If the check fails, the function shall raise the development error `SOAD_E_NOTINIT` and return `E_NOT_OK`. ⌋ ( )

[SOAD221] ⌈
If development error detection is enabled: the function shall check parameter `PbufPtr` for being a `NULL_PTR`. If this is `TRUE`, the function shall raise the development error `SOAD_E_NULL_PTR` and return `E_NOT_OK`. ⌋ ( )

[SOAD252] ⌈
If development error detection is enabled: the function shall check parameter `PbufPtr` for being valid. If the check fails, the function shall raise the development error `SOAD_E_PARAM_POINTER` and return `E_NOT_OK`. ⌋ ( )

[SOAD253] ⌈
If development error detection is enabled: the function shall check parameter `SoHandle` for being valid. If the check fails, the function shall raise the development error `SOAD_E_INVAL` and return `E_NOT_OK`. ⌋ ( )

[SOAD254] ⌈
If development error detection is enabled: the function shall check parameter `Destination` for being valid. If the check fails, the function shall raise the development error `SOAD_E_INVAL` and return `E_NOT_OK`. ⌋ ( )

[SOAD255] ⌈
If development error detection is enabled: the function shall check parameter `Length` for being valid. If the check fails, the function shall raise the development error `SOAD_E_INVAL` and return `E_NOT_OK`. ⌋ ( )

8.3.5.2 TcpIp_TransmitTo

[SOAD085] ⌈

Service name:	TcpIp_TransmitTo	
Syntax:	`Std_ReturnType TcpIp_TransmitTo( int SoHandle, SoAd_TcpIpPbufType* PbufPtr, SoAd_TcpIp_IpAddrPortType Destination )`	
Service ID[hex]:	0x82	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant	
Parameters (in):	SoHandle	This parameter contains the socket handle on which the data is to be transmitted.
	PbufPtr	Pointer to the payload data to be transmitted.
	Destination	IP address and port where data is to be sent to.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The request has been accepted E_NOT_OK: The request has not been accepted, e.g. due to a still ongoing transmission in the corresponding socket or the to be transmitted message is too long or the socket is not connected (for TCP).
Description:	This service is utilized to request the transfer of data. The Buffer to which Data points must not be modified until the transmission indication was received.	

⌋( )

[SOAD251] ⌈
If development error detection is enabled: the function shall check that the service `TcpIp_Init` was previously called. If the check fails, the function shall raise the development error `SOAD_E_NOTINIT` and return `E_NOT_OK`. ⌋( )

[SOAD256] ⌈
If development error detection is enabled: the function shall check parameter `PbufPtr` for being a `NULL_PTR`. If this is `TRUE`, the function shall raise the development error `SOAD_E_NULL_PTR` and return `E_NOT_OK`. ⌋( )

[SOAD257] ⌈
If development error detection is enabled: the function shall check parameter `PbufPtr` for being valid. If the check fails, the function shall raise the development error `SOAD_E_PARAM_POINTER` and return `E_NOT_OK`. ⌋( )

[SOAD258] ⌈
If development error detection is enabled: the function shall check parameter `SoHandle` for being valid. If the check fails, the function shall raise the development error `SOAD_E_INVAL` and return `E_NOT_OK`. ⌋( )

[SOAD259] ⌈
If development error detection is enabled: the function shall check parameter `Destination` for being valid. If the check fails, the function shall raise the development error `SOAD_E_INVAL` and return `E_NOT_OK`. ⌋( )

8.3.5.3 TcpIp_Received

[SOAD086] ⌈

Service name:	TcpIp_Received	

Syntax:	`Std_ReturnType TcpIp_Received(`
	`  int SoHandle,`
	`  uint32 Length`
	`)`	

Service ID[hex]:	0x83	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant	
Parameters (in):	SoHandle	This parameter contains the socket handler related to this call.
	Length	Number of bytes to be freed in the receive buffer.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The request has been accepted E_NOT_OK: The request has not been accepted
Description:	By this API service the reception of socket data is confirmed to the TCP/IP stack. The TCP/IP stack shall free allocated buffers. The TCP/IP will increase the advertised TCP window in case of a TCP connection.	

⌋ ( )

[SOAD241] ⌈
If development error detection is enabled: the function shall check that the service `TcpIp_Init` was previously called. If the check fails, the function shall raise the development error `SOAD_E_NOTINIT` and return `E_NOT_OK`. ⌋ ( )

[SOAD242] ⌈
If development error detection is enabled: the function shall check parameter `SoHandle` for being valid. If the check fails, the function shall raise the development error `SOAD_E_INVAL` and return `E_NOT_OK`. ⌋ ( )

[SOAD260] ⌈
If development error detection is enabled: the function shall check parameter `Length` for being valid. If the check fails, the function shall raise the development error `SOAD_E_INVAL` and return `E_NOT_OK`. ⌋ ( )

8.3.5.4 TcpIp_TcpConnect

[SOAD087] ⌈

Service name:	TcpIp_TcpConnect	

Syntax:	`Std_ReturnType TcpIp_TcpConnect(`
	`  int SoHandle,`
	`  SoAd_TcpIp_IpAddrPortType Destination`
	`)`	

Service ID[hex]:	0x84	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant	
Parameters (in):	SoHandle	This parameter contains the socket handler of the TCP connection which shall be established.
	Destination	IP address and port to be connected to.

Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The request has been accepted E_NOT_OK: The request has not been accepted: the socket is not configured to be a client socket.
Description:	By this API service the TCP/IP stack is requested to establish a TCP connection to the configured peer.	

⌋ ( )

[SOAD243] ⌈
If development error detection is enabled: the function shall check that the service `TcpIp_Init` was previously called. If the check fails, the function shall raise the development error `SOAD_E_NOTINIT` and return `E_NOT_OK`. ⌋ ( )

[SOAD244] ⌈
If development error detection is enabled: the function shall check parameter `SoHandle` for being valid. If the check fails, the function shall raise the development error `SOAD_E_INVAL` and return `E_NOT_OK`. ⌋ ( )

[SOAD261] ⌈
If development error detection is enabled: the function shall check parameter `Destination` for being valid. If the check fails, the function shall raise the development error `SOAD_E_INVAL` and return `E_NOT_OK`. ⌋ ( )

8.3.5.5 TcpIp_Listen

[SOAD088] ⌈

Service name:	TcpIp_Listen	

Syntax:	`Std_ReturnType TcpIp_Listen(`
	`  int SoHandle`
	`)`	

Service ID[hex]:	0x85	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant	
Parameters (in):	SoHandle	This parameter contains the socket handle of the TCP/UDP connection which shall be put into the listen state.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The request has been accepted E_NOT_OK: The request has not been accepted, the socket is not configured to be a server socket.
Description:	By this API service the TCP/IP stack is requested to listen to the TCP or UDP port specified in the socket handle.	

⌋ ( )

[SOAD245] ⌈
If development error detection is enabled: the function shall check that the service `TcpIp_Init` was previously called. If the check fails, the function shall raise the development error `SOAD_E_NOTINIT` and return `E_NOT_OK`. ⌋ ( )

[SOAD246] ⌈
If development error detection is enabled: the function shall check parameter `SoHandle` for being valid. If the check fails, the function shall raise the development error `SOAD_E_INVAL` and return `E_NOT_OK`. ⌋ ( )

8.3.5.6 TcpIp_TcpClose

[SOAD089] ⌈

Service name:	TcpIp_TcpClose	
Syntax:	`Std_ReturnType TcpIp_TcpClose( int SoHandle )`	
Service ID[hex]:	0x86	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant	
Parameters (in):	SoHandle	This parameter contains the socket handle of the TCP connection which shall be closed.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The request has been accepted E_NOT_OK: The request has not been accepted, the TCP connection was not established before.
Description:	By this API service the TCP/IP stack is requested to close a TCP connection. If the connection is active a FIN segment is sent to the peer. If the socket is in the Listen state, the Listen state will be left.	

⌋ ( )

[SOAD247] ⌈
If development error detection is enabled: the function shall check that the service `TcpIp_Init` was previously called. If the check fails, the function shall raise the development error `SOAD_E_NOTINIT` and return `E_NOT_OK`. ⌋ ( )

[SOAD248] ⌈
If development error detection is enabled: the function shall check parameter `SoHandle` for being valid. If the check fails, the function shall raise the development error `SOAD_E_INVAL` and return `E_NOT_OK`. ⌋ ( )

8.3.5.7 TcpIp_ChangeParameter

[SOAD090] ⌈

Service name:	TcpIp_ChangeParameter	
Syntax:	`Std_ReturnType TcpIp_ChangeParameter( int SoHandle, uint8 ParameterId, sint32 ParameterValue )`	
Service ID[hex]:	0x87	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	SoHandle	This parameter contains the socket handle of the TCP connection which is to be configured.
	ParameterId	Identifier of the parameter to be changed
	ParameterValue	New value of the parameter to be set
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	E_OK: The request has been accepted E_NOT_OK: The request has not been accepted
Description:	By this API service the TCP/IP stack is requested to change a connection parameter. E.g. the Nagle algorithm may be controlled by this API.	

⌋ ( )

[SOAD249] ⌈
If development error detection is enabled: the function shall check that the service `TcpIp_Init` was previously called. If the check fails, the function shall raise the development error `SOAD_E_NOTINIT` and return `E_NOT_OK`. ⌋ ( )

[SOAD250] ⌈
If development error detection is enabled: the function shall check parameter `SoHandle` for being valid. If the check fails, the function shall raise the development error `SOAD_E_INVAL` and return `E_NOT_OK`. ⌋ ( )

[SOAD263] ⌈
If development error detection is enabled: the function shall check parameter `ParameterId` for being valid. If the check fails, the function shall raise the development error `SOAD_E_INVAL` and return `E_NOT_OK`. ⌋ ( )

[SOAD262] ⌈
If development error detection is enabled: the function shall check parameter `ParameterValue` for being valid. If the check fails, the function shall raise the development error `SOAD_E_INVAL` and return `E_NOT_OK`. ⌋ ( )

8.4 Call-back notifications

In AUTOSAR, the functions a module provides to layers which are placed below the module in the AUTOSAR software layer model, are called 'call-back functions'. Generally, a software entity A (SoAd), which, in order to be informed about some event C in software entity B (TCP/IP stack), is registered as interested in event C at

software entity B by calling a register mechanism B provides, and is called by entity B if event C occurs. In AUTOSAR the Call-back is usually implicitly registered by configuration.

[SOAD124] ⌈The function prototypes of the call-back functions shall be provided in the file `SoAd_Cbk.h` (see 5.1.2).⌋( )

The following services of the SoAd are called by the TCP/IP Stack if the Call-back socket API is used.

No call-back notifications are present if only the BSD socket (COTS) interface is implemented.

8.4.1 SoAd_TcpIpRxIndication

[SOAD097] ⌈

Service name:	SoAd_TcpIpRxIndication	
Syntax:	`void SoAd_TcpIpRxIndication( int SoHandle, SoAd_TcpIpPbufType* PbufPtr, SoAd_TcpIp_IpAddrPortType Source )`	
Service ID[hex]:	0x05	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	SoHandle	This parameter contains the socket handle of the received data.
	PbufPtr	Pointer to data pool buffer.
	Source	IP address and port where the received data was sent from.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	The TCP/IP stack calls this primitive after the reception of data on a socket. The socket handle along with configuration information determins which module or plug-in is to be called.	

⌋( )

[SOAD264] ⌈
If development error detection is enabled: the function shall check that the service `SoAd_Init` was previously called. If the check fails, the function shall raise the development error `SOAD_E_NOTINIT`.⌋( )

[SOAD265] ⌈
If development error detection is enabled: the function shall check parameter `PbufPtr` for being a `NULL_PTR`. If this is `TRUE`, the function shall raise the development error `SOAD_E_NULL_PTR` and return `E_NOT_OK`.⌋( )

[SOAD266] ⌈

If development error detection is enabled: the function shall check parameter PbufPtr for being valid. If the check fails, the function shall raise the development error SOAD_E_PARAM_POINTER and return E_NOT_OK. ⌋( )

[SOAD267] ⌈
If development error detection is enabled: the function shall check parameter SoHandle for being valid. If the check fails, the function shall raise the development error SOAD_E_INVAL and return E_NOT_OK. ⌋( )

[SOAD268] ⌈
If development error detection is enabled: the function shall check parameter Source for being valid. If the check fails, the function shall raise the development error SOAD_E_INVAL and return E_NOT_OK. ⌋( )

8.4.2 SoAd_TcpIpTxConfirmation

[SOAD098] ⌈

Service name:	SoAd_TcpIpTxConfirmation	

Syntax:	`void SoAd_TcpIpTxConfirmation(`
	`  int SoHandle,`
	`  uint32 Length`
	`)`	

Service ID[hex]:	0x06	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	SoHandle	This parameter contains the socket handle of the socket the data is to be transmitted on.
	Length	Number of transmitted data bytes.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	The TCP/IP stack calls this function after the data has been acknowledged by the peer for TCP or was sent to the lower layer driver using UDP. Caveats: The upper layer might not be able to determine exactly which data bytes have been confirmed.	

⌋( )

[SOAD269] ⌈
If development error detection is enabled: the function shall check that the service SoAd_Init was previously called. If the check fails, the function shall raise the development error SOAD_E_NOTINIT. ⌋( )

[SOAD270] ⌈
If development error detection is enabled: the function shall check parameter SoHandle for being valid. If the check fails, the function shall raise the development error SOAD_E_INVAL and return E_NOT_OK. ⌋( )

[SOAD271] ⌈
If development error detection is enabled: the function shall check parameter `Length` for being valid. If the check fails, the function shall raise the development error `SOAD_E_INVAL` and return `E_NOT_OK`. ⌋ ( )

8.4.3 SoAd_TcpAccepted

[SOAD099] ⌈

Service name:	SoAd_TcpAccepted	

Syntax:	`void SoAd_TcpAccepted(`
	`  int SoHandle`
	`)`	

Service ID[hex]:	0x0C	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	SoHandle	This parameter contains the socket handle of the socket that has been connected.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	SoAd_TcpAccepted() gets called if the stack put a socket into the listen mode before (as server) and a peer connected to it (as client). In detail: The TCP/IP stack calls this function after a socket was set into the listen state with TcpIp_Listen() and a TCP connection is requested by the peer. The parameter value of SoHandle equals the SoHandle value of the preceeding TcpIp_Listen() call.	

⌋ ( )

[SOAD272] ⌈
If development error detection is enabled: the function shall check that the service `SoAd_Init` was previously called. If the check fails, the function shall raise the development error `SOAD_E_NOTINIT`. ⌋ ( )

[SOAD273] ⌈
If development error detection is enabled: the function shall check parameter `SoHandle` for being valid. If the check fails, the function shall raise the development error `SOAD_E_INVAL` and return `E_NOT_OK`. ⌋ ( )

8.4.4 SoAd_TcpConnected

[SOAD100] ⌈

Service name:	SoAd_TcpConnected	

Syntax:	`void SoAd_TcpConnected(`
	`  int SoHandle`
	`)`	

Service ID[hex]:	0x0D	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	SoHandle	This parameter contains the socket handle of the socket that has been connected.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	SoAd_TcpConnected() gets called if the stack initiated a TCP connection before (as client) and the peer (the server) acknowledged the connection set up. In detail: The TCP/IP stack calls this function after a socket was requested to connect with TcpIp_TcpConnect() and a TCP connection is confirmed by the peer. The parameter value of SoHandle equals the SoHandle value of the preceeding TcpIp_TcpConnect() call.	

⌋( )

[SOAD274] ⌈
If development error detection is enabled: the function shall check that the service `SoAd_Init` was previously called. If the check fails, the function shall raise the development error `SOAD_E_NOTINIT`. ⌋( )

[SOAD275] ⌈
If development error detection is enabled: the function shall check parameter `SoHandle` for being valid. If the check fails, the function shall raise the development error `SOAD_E_INVAL` and return `E_NOT_OK`. ⌋( )

8.4.5 SoAd_TcpIpEvent

[SOAD146] ⌈

Service name:	SoAd_TcpIpEvent	

Syntax:	`void SoAd_TcpIpEvent(`
	`  int SoHandle,`
	`  SoAd_TcpIpEventType Event`
	`)`	

Service ID[hex]:	0x0E	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	SoHandle	This parameter contains the socket handle of the socket the data is to be transmitted on.
	Event	This parameter contains a description of the event just encountered.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	SoAd_TcpIpEvent() gets called if the stack encounters a condition described by the values in TcpIpEvent.	

⌋( )

[SOAD276] ⌈
If development error detection is enabled: the function shall check that the service SoAd_Init was previously called. If the check fails, the function shall raise the development error SOAD_E_NOTINIT. ⌋ ( )

[SOAD277] ⌈
If development error detection is enabled: the function shall check parameter SoHandle for being valid. If the check fails, the function shall raise the development error SOAD_E_INVAL and return E_NOT_OK. ⌋ ( )

[SOAD278] ⌈
If development error detection is enabled: the function shall check parameter Event for being valid. If the check fails, the function shall raise the development error SOAD_E_INVAL and return E_NOT_OK. ⌋ ( )

8.4.6 SoAd_Cbk_LocalIpAssignmentChg

[SOAD209] ⌈

Service name:	SoAd_Cbk_LocalIpAssignmentChg	

Syntax:	`void SoAd_Cbk_LocalIpAssignmentChg(`
	`  uint8 Index,`
	`  boolean Valid,`
	`  SoAd_SockAddrType Address`
	`)`	

Service ID[hex]:	0x12	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	Index	This parameter contains the IP interface index (to be chosen by the IP stack).
	Valid	This parameter is TRUE if a valid IP address is assigned, otherwise FALSE.
	Address	This Parameter contains the new valid IP address, if Valid is TRUE, else it contains the invalidated IP address, or 0.0.0.0, if no address was ever assigned.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	SoAd_Cbk_LocalIpAssignmentChg() gets called by the TCP/IP stack if an IP address changes (i.e. new address assigned or assigned address becomes invalid). In most cases, where one IP Address is bound to the physical contoller interface, the Index can be the controller index. In order to allow for the TCP/IP stack to maintain multiple IP addresses on one single physical contoller interface the index was left open to be chosen by the stack.	

⌋ ( )

[SOAD279] ⌈
If development error detection is enabled: the function shall check that the service SoAd_Init was previously called. If the check fails, the function shall raise the development error SOAD_E_NOTINIT. ⌋ ( )

[SOAD280] ⌈
If development error detection is enabled: the function shall check parameter `Index` for being valid. If the check fails, the function shall raise the development error `SOAD_E_INVAL` and return `E_NOT_OK`. ⌋ ( )

[SOAD281] ⌈
If development error detection is enabled: the function shall check parameter `Valid` for being valid. If the check fails, the function shall raise the development error `SOAD_E_INVAL` and return `E_NOT_OK`. ⌋ ( )

[SOAD282] ⌈
If development error detection is enabled: the function shall check parameter `Address` for being valid. If the check fails, the function shall raise the development error `SOAD_E_INVAL` and return `E_NOT_OK`. ⌋ ( )

8.4.7 SoAd_BusSM_ModeIndication

[SOAD285] ⌈

Service name:	SoAd_BusSM_ModeIndication	
Syntax:	`void SoAd_BusSM_ModeIndication( NetworkHandleType Channel, ComM_ModeType* ComModePtr )`	
Service ID[hex]:	0x13	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant	
Parameters (in):	Channel	Identifies the communication medium.
	ComModePtr	Status of the ComM state machine for this Channel.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Indication of the bus mode by EthSM. SoAd shall trigger initialization of the TCP/IP stack, if ComModePtr is COMM_FULL_COMMUNICATION. Upon COMM_NO_COMMUNICATION the SoAd shall shut down the TCP/IP stack.	

⌋ ( )

[SOAD286] ⌈
If development error detection is enabled: the function shall check that the service `SoAd_Init` was previously called. If the check fails, the function shall raise the development error `SOAD_E_NOTINIT`. ⌋ ( )

[SOAD288] ⌈
If development error detection is enabled: the function shall check parameter `Channel` for being valid. If the check fails, the function shall raise the development error `SOAD_E_INVAL` and return `E_NOT_OK`. ⌋ ( )

[SOAD289] ⌈

If development error detection is enabled: the function shall check parameter ComModePtr for being a NULL_PTR. If this is TRUE, the function shall raise the development error SOAD_E_NULL_PTR and return E_NOT_OK. ⌋( )

[SOAD290] ⌈
If development error detection is enabled: the function shall check parameter ComModePtr for being valid. If the check fails, the function shall raise the development error SOAD_E_PARAM_POINTER and return E_NOT_OK. ⌋( )

8.5 Scheduled functions

These functions are directly called by Basic Software Scheduler. The following functions shall have no return value and no parameter. All functions shall be non reentrant.

8.5.1 Terms and definitions

Fixed cyclic: The term fixed cyclic means that a specific configured cycle time shall not change during runtime, as the functionality requires that fixed timing.

Variable cyclic: Variable cyclic means that the cycle times are defined at configuration, but might be mode dependent and therefore vary during runtime.

On pre condition: On pre condition means that no cycle time can be defined. The function will be called when conditions are fulfilled. Alternatively, the function may be called cyclically however the cycle time will be assigned dynamically during runtime by other modules.

8.5.2 SoAd_MainFunction

[SOAD121] ⌈

Service name:	SoAd_MainFunction

Syntax:	`void SoAd_MainFunction(`
	`  void`
	`)`

Service ID[hex]:	0x10
Timing:	FIXED_CYCLIC
Description:	Schedules the Socket Adaptor. (Entry point for scheduling)

⌋( )

[SOAD131] ⌈
The main function for scheduling the SoAd (Entry point for scheduling) shall be called by the Schedule Manager according to the configured call period. ⌋( )

[SOAD176] ⌈
The call period of the SoAd_MainFunction() is determined by configuration parameter SOAD_MAINFUNCTION_PERIOD. ⌋( )

[SOAD283] ⌈
If development error detection is enabled: the function shall check that the service `SoAd_Init` was previously called. If the check fails, the function shall raise the development error `SOAD_E_NOTINIT`. ⌋ ( )

8.5.3 TcpIp_MainFunctionCyclic

[SOAD143] ⌈

Service name:	TcpIp_MainFunctionCyclic

Syntax:	`void TcpIp_MainFunctionCyclic(`
	`  void`
	`)`

Service ID[hex]:	0x8B
Timing:	FIXED_CYCLIC
Description:	Schedules the TCP/IP stack. (Entry point for scheduling)

⌋ ( )

[SOAD177] ⌈
The main function for scheduling the TCP/IP stack (Entry point for scheduling) shall be called by the Schedule Manager according to the configured call period. ⌋ ( )

[SOAD178] ⌈
The call period of the `TcpIp_MainFunctionCyclic()` is determined by configuration parameter `SOAD_TCPIP_MAINFUNCTION_PERIOD`. ⌋ ( )

[SOAD284] ⌈
If development error detection is enabled: the function shall check that the service `TcpIp_Init` was previously called. If the check fails, the function shall raise the development error `SOAD_E_NOTINIT`. ⌋ ( )

8.6 Expected Interfaces of the SoAd

In this chapter all interfaces required by the SoAd from other modules are listed.
Some of the interfaces listed here are defined in this document as this document also specifies the TCP/IP stacks interfaces.

8.6.1 Mandatory Interfaces of the SoAd

This chapter defines all interfaces which are required by the SoAd to fulfill the core functionality of the SoAd module.

API function	*Description*
Dem_ReportErrorStatus	Queues the reported events from the BSW modules (API is only used by BSW modules). The interface has an asynchronous behavior, because the processing of the event is done within the Dem main function.
PduR_SoAdCopyRxData	This function is called when a transport protocol module has data to copy for the receiving module. Several calls may be made during one transportation of an I-PDU. The service shall provide the currently available buffer size when invoked with info.SduLength equal to 0.
PduR_SoAdCopyTxData	This function is called by the transport protocol module to query the transmit data of an I-PDU segment. Each call to this function copies the next part of the transmit data until TpDataState indicates TP_DATARETRY. In this case the API restarts to copy the data beginning at the location indicated by TpTxDataCnt. The service shall provide the size of the remaining data when invoked with info.SduLength equal to 0.
PduR_SoAdRxIndication	Called by the transport protocol module after an I-PDU has been received successfully or when an error occurred. It is also used to confirm cancellation of an I-PDU.
PduR_SoAdStartOfReception	This function will be called by the transport protocol module at the start of receiving an I-PDU. The I-PDU might be fragmented into multiple N-PDUs (FF with one or more following CFs) or might consist of a single N-PDU (SF). The service shall provide the currently available maximum buffer size when invoked with TpSduLength equal to 0.
PduR_SoAdTxConfirmation	This function is called by a transport protocol module after the I-PDU has been transmitted on its network, the result will reveal if the transmission was successful or not.
PduR_SoAdTriggerTransmit	The lower layer communication module requests the buffer of the SDU for transmission from the upper layer module.

8.6.2 Mandatory Interfaces of the DoIP plug-in

8.6.2.1 <user>_SoAdGetVin

[SOAD157] ⌈

Service name:	<User>_SoAdGetVin	

Syntax:	`void <User>_SoAdGetVin(`
	`  uint8** DataPtr`
	`)`	

Service ID[hex]:	0x00	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	None	
Parameters (inout):	DataPtr	Pointer to the data structure for the Vehicle Identification Number of 17 bytes length.
Parameters (out):	None	
Return value:	None	
Description:	Returns the Vehicle Identification Number.	

⌋ ( )

8.6.2.2 EthIf_GetPhysAddr

[SOAD158] ⌈

Service name:	EthIf_GetPhysAddr	

Syntax:	`void EthIf_GetPhysAddr(`
	`  uint8 CtrlIdx,`
	`  uint8* PhysAddrPtr`
	`)`	

Service ID[hex]:	0x08	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	CtrlIdx	Index of the Ethernet controller within the context of the Ethernet Interface
Parameters (inout):	None	
Parameters (out):	PhysAddrPtr	Physical source address (MAC address) in network byte order. Please refer to [16] for the physical source address specification.
Return value:	None	
Description:	Obtains the physical source address used by the indexed controller	

⌋ ( )

8.6.3 Optional Interfaces of the SoAd

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

One of the APIs, either BSD Socket or Call-back is required!

API function	*Description*
SoAd_SoAdGetVin	Returns the Vehicle Identification Number.
DoIP_GetVersionInfo	Returns the version information.
PduR_SoAdRxIndication	Indication of a received I-PDU from a lower layer communication module.
PduR_SoAdTxConfirmation	The lower layer communication module confirms the transmission of an I-PDU.
TcpIp_ChangeParameter	By this API service the TCP/IP stack is requested to change a connection parameter. E.g. the Nagle algorithm may be controlled by this API.
TcpIp_GetVersionInfo	Returns the version information.
TcpIp_Init	This service initializes the TCP/IP Stack. TcpIp_Init may not block the start-up process for an indefinite amount of time. Caveats: The call of this service is mandatory before using the TcpIp instance for further processing.
TcpIp_Listen	By this API service the TCP/IP stack is requested to listen to the TCP or UDP port specified in the socket handle.
TcpIp_MainFunctionCyclic	Schedules the TCP/IP stack. (Entry point for scheduling)
TcpIp_ProvideTxBuffer	This service is utilized to request the transfer of data. The Buffer to which Data points must not be modified until the transmission confirmation was received.
TcpIp_Received	By this API service the reception of socket data is confirmed to the TCP/IP stack. The TCP/IP stack shall free allocated buffers. The TCP/IP will increase the advertised TCP window in case of a TCP connection.
TcpIp_SetDhcpHostNameOption	This API sets the DHCP Host Name Option according to ISO 13400. The DHCP Host Name Option may consist of static and dynamic content. The static content will usually be found in SOAD053_Conf: SoAdDoIpHostNameOpt. This API needs to be implemented whenever DoIP is to be supported, independent of the API used.
TcpIp_SetDhcpHostNameOption	This API sets the DHCP Host Name Option according to ISO 13400. The DHCP Host Name Option may consist of static and dynamic content. The static content will usually be found in SOAD053_Conf: SoAdDoIpHostNameOpt. This API needs to be implemented whenever DoIP is to be supported, independent of the API used.
TcpIp_Shutdown	This service closes all pending transport protocol connections, releases all resources and stops the TCP/IP stack.
TcpIp_TcpClose	By this API service the TCP/IP stack is requested to close a TCP connection. If the connection is active a FIN segment is sent to the peer. If the socket is in the Listen state, the Listen state will be left.
TcpIp_TcpConnect	By this API service the TCP/IP stack is requested to establish a TCP connection to the configured peer.
TcpIp_TransmitTo	This service is utilized to request the transfer of data. The Buffer to which Data points must not be modified until the transmission indication was received.
UdpNm_SoAdIfRxIndication	This service indicates a successful reception of a received NM message to the UdpNm after passing all filters and validation checks. Caveats: - Until this service returns the SoAd will not access udpSduPtr. The udpSduPtr is only valid and can be used by upper layers until the indication returns. SoAd guarantees that the number of configured bytes for this udpNmRxPduId is valid. The call context is either on

	interrupt level (interrupt mode) or on task level (polling mode). - The UdpNm module is initialized correctly.
UdpNm_SoAdIfTxConfirmation	This service confirms a previous successfully processed transmit request. Caveats: - The call context is either on interrupt level (interrupt mode) or on task level (polling mode). - The UdpNm module is initialized correctly.
Xcp_<module>RxIndication	This function is called by the lower layers (i.e. FlexRay Interface, TTCAN Interface and Socket Adaptor or CDD) when an AUTOSAR XCP PDU has been received
Xcp_<module>TxConfirmation	This function is called by the lower layers (i.e. FlexRay Interface, TTCAN Interface and Socket Adaptor or CDD) when an AUTOSAR XCP PDU has been transmitted
Xcp_SoAdTriggerTransmit	The lower layer communication module requests the buffer of the SDU for transmission from the upper layer module.
accept	The accept function shall extract the first connection on the queue of pending connections, create a new socket with the same socket type protocol and address family as the specified socket, and allocate a new file descriptor for that socket.
bind	The bind function shall assign the local socket address specified in address to a socket identified by descriptor socket that has no local socket address assigned. Sockets created with the socket function are initially unnamed; they are identified only by their address family.
close	The close function shall de-allocate the file descriptor indicated by socket.
connect	If the socket has not already been bound to a local address, connect shall bind it to an address which is an unused local address. If the initiating socket is not connection-mode, then connect shall set the sockets peer address, and no connection is made. For SOCK_DGRAM sockets, the peer address identifies where all datagrams are sent on subsequent send functions. If address is a null address for the protocol, the sockets peer address shall be reset. If the connection cannot be established immediately and O_NONBLOCK is set for the file descriptor for the socket, connect() shall fail [EINPROGRESS], but the connection request shall not be aborted, and the connection shall be established asynchronously. Subsequent calls to connect() for the same socket, before the connection is established, shall fail [EALREADY]. When the connection has been established asynchronously, select() and poll() shall indicate that the file descriptor for the socket is ready for writing. Note: For non-blocking calls [O_NONBLOCK], a failure is the normal response.
fcntl	Send commands to the TCP/IP stack for configuration. Or read status of configuration flags.
fcntl	Send commands to the TCP/IP stack for configuration. Or read status of configuration flags.
getlasterror	This function returns the last network error that occurred.
listen	The listen function shall mark a connection-mode socket, specified by the socket argument, as accepting connections.
poll	The poll() function provides SoAd with a mechanism for multiplexing input/output over a set of file descriptors. For each member of the array pointed to by fds, poll() shall examine the given file descriptor for the event(s) specified in events. The number of SoAd_pollfdType structures in the fds array is specified by nfds. The poll() function shall identify those file descriptors on which an application can read or write data, or on which certain events have occurred.
recvfrom	The recvfrom function shall receive a message from a connection-

	mode or connectionless-mode socket. Caveats: For message-based sockets, such as SOCK_DGRAM, the entire message shall be read in a single operation. If a message is too long to fit in the supplied buffer, and MSG_PEEK is not set in the flags argument, the excess bytes shall be discarded. For stream-based sockets, such as SOCK_STREAM, message boundaries shall be ignored, and no data shall be discarded.
sendto	The sendto() function shall send a message through a connection-mode or connectionless-mode socket. If the socket is connectionless-mode, the message shall be sent to the address specified by dest_addr. If the socket is connection-mode, dest_addr shall be ignored.
setsockopt	The setsockopt function shall set the option specified by the option_name argument, at the protocol level specified by the level argument, to the value pointed to by the option_value argument for the socket associated with the file descriptor specified by the socket argument.
socket	The socket function shall create an unbound socket in a communications domain, and return a file descriptor (called socket) that can be used in later function calls that operate on sockets.

8.6.4 Configurable interfaces of the SoAd

In this chapter all interfaces are listed, where the target function of any upper layer to be called has to be set up by configuration. These call-out services are specified and implemented in the upper communication modules, which use the TCP/IP stack according to the AUTOSAR Socket API. The specific call-out notification is specified in the corresponding SWS documents.

As far the interface name is not specified to be mandatory, no call-out is performed, if no API name is configured. This chapter describes only the content of notification of the call-out, the call context inside the TCP/IP stack and exact time by the call event.

`<User>_NotificationName` - This condition is applied for such interface services which will be implemented in the upper layer ('user') and called by the TCP/IP stack. This condition displays the symbolic name of the functional group in a call-out service in the corresponding upper layer. Each upper layer can define none, one, or several call-out services for the same functionality (i.e. transmit confirmation). The dispatch is ensured by the socket handle and the Socket Connection Table.

8.6.4.1 <User>_SoAdIfRxIndication (PduR, UdpNm, Xcp, CDD)

[SOAD106] ⌈

Service name:	<User_SoAdIfRxIndication>	

Syntax:	`void <User_SoAdIfRxIndication>(`
	`  PduIdType RxPduId,`
	`  PduInfoType* PduInfoPtr`
	`)`	

Sync/Async:	Synchronous	
Reentrancy:	Reentrant for different PduIds. Non reentrant for the same PduId.	
Parameters (in):	RxPduId	ID of the received I-PDU.
	PduInfoPtr	Contains the length (SduLength) of the received I-PDU and a pointer to a buffer (SduDataPtr) containing the I-PDU.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Indication of a received I-PDU from a lower layer communication module.	

⌋ ( )

8.6.4.2 <User>_SoAdIfTxConfirmation (PduR, UdpNm, Xcp, CDD)

[SOAD107] ⌈

Service name:	<User_SoAdIfTxConfirmation>	

Syntax:	`void <User_SoAdIfTxConfirmation>(`
	`  PduIdType TxPduId`
	`)`	

Sync/Async:	Synchronous	
Reentrancy:	Reentrant for different PduIds. Non reentrant for the same PduId.	
Parameters (in):	TxPduId	ID of the I-PDU that has been transmitted.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	The lower layer communication module confirms the transmission of an I-PDU.	

⌋ ( )

8.6.4.3 <User>_SoAdIfTriggerTransmit

[SOAD188] ⌈The SoAd shall not use the API `<User>_SoAdIfTriggerTransmit.` ⌋ ( )

If the underying TCP/IP stack supports immediate transmit.

8.6.4.4 <User>_SoAdTpCopyRxData (PduR, CDD)

[SOAD139] ⌈

Service name:	<User_SoAdTpCopyRxData>	

Syntax:	`BufReq_ReturnType <User_SoAdTpCopyRxData>(`
	`  PduIdType RxPduId,`
	`  PduInfoType* PduInfoPtr,`
	`  PduLengthType* bufferSizePtr`
	`)`	

Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	RxPduId	Identification of the received I-PDU.
	PduInfoPtr	Pointer to the buffer (SduDataPtr) and its length (SduLength) containing the data to be copied by PDU Router module in case of gateway or upper layer module in case of reception.
Parameters (inout):	None	
Parameters (out):	bufferSizePtr	Available receive buffer after data has been copied.
Return value:	BufReq_ReturnType	BUFREQ_OK: Buffer request accomplished successful. BUFREQ_E_NOT_OK: Buffer request not successful. Buffer cannot be accessed. BUFREQ_E_BUSY: Temporarily no buffer available. It's up the requestor to retry request for a certain time.
Description:	This function is called when transport protocol module have data to copy to the receiving module. Several calls may be made during one transportation of an I-PDU.	

⌋ ( )

8.6.4.5 <User>_SoAdTpRxIndication (PduR, CDD)

[SOAD180] ⌈

Service name:	<User_SoAdTpRxIndication>	

Syntax:	`void <User_SoAdTpRxIndication>(`
	`  PduIdType RxPduId,`
	`  NotifResultType result`
	`)`	

Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	RxPduId	Identification of the received I-PDU.
	result	Result of the reception.
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Called by the transport protocol module after an I-PDU has been received successfully or when an error occurred. It is also used to confirm cancellation of an I-PDU.	

⌋ ( )

8.6.4.6 <User>_SoAdTpStartofReception (PduR, CDD)

[SOAD138] ⌈

Service name:	<User_SoAdTpStartofReception>	
Syntax:	`BufReq_ReturnType <User_SoAdTpStartofReception>( PduIdType RxPduId, PduLengthType TpSduLength, PduLengthType* bufferSizePtr )`	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	RxPduId	Identification of the I-PDU.
	TpSduLength	Total length of the PDU to be received.
Parameters (inout):	None	
Parameters (out):	bufferSizePtr	Available receive buffer in the receiving module. This parameter will be used to compute Block Size (BS) in the transport protocol module.
Return value:	BufReq_ReturnType	BUFREQ_OK: Connection has been accepted. RxBufferSizePtr indicates the available receive buffer. BUFREQ_E_BUSY: Currently no buffer of the requested size is available. RxBufferSizePtr remains unchanged. Connection has been rejected. BUFREQ_E_NOT_OK: Connection has been rejected. RxBufferSizePtr remains unchanged. BUFREQ_E_OVFL: No Buffer of the required length can be provided.
Description:	This function will be called by the transport protocol module at the start of receiving an I-PDU. The I-PDU might be fragmented into multiple N-PDUs (FF with one or more following CFs) or might consist of a single N-PDU (SF). The service shall provide the currently available maximum buffer size when invoked with TpSdulength equal to 0.	

⌋ ( )

8.6.4.7 <User>_SoAdTpCopyTxData (PduR, CDD)

[SOAD137] ⌈

Service name:	<User_SoAdTpCopyTxData>	
Syntax:	`BufReq_ReturnType <User_SoAdTpCopyTxData>( PduIdType TxPduId, PduInfoType* PduInfoPtr, RetryInfoType* retry, PduLengthType availableDataPtr )`	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	TxPduId	Identification of the transmitted I-PDU.
	PduInfoPtr	Provides destination buffer and the number of bytes to copy. In case of gateway the PDU Router module will copy otherwise the source upper layer module will copy the data. If no enough transmit data is available, no data is copied. The transport protocol module will retry. A size of copy size of 0 can be used to indicate state changes in the retry parameter.
	retry	This parameter is used to retransmit data because problems occurred in transmitting it the last time. If the I-PDU is transmitted from a local module (e.g. DCM) the PDU router module will just forward the parameter value without check. If the I-PDU is gatewayed from another bus the

		PDU Router module will make the following interpretation: If the transmitted TP I-PDU does not support the retry feature a NULL_PTR can be provided. This indicates that the copied transmit data can be removed from the buffer after it has been copied. If the retry feature is used by the Tx I-PDU, RetryInfoPtr must point to a valid RetryInfoType element. If TpDataState indicates TP_CONFPENDING the previously copied data must remain in the TP buffer to be available for error recovery. TP_DATACONF indicates that all data, that have been copied so far, are confirmed and can be removed from the TP buffer. Data copied by this API call are excluded and will be confirmed later. TP_DATARETRY indicates that this API call shall copy already copied data in order to recover from an error. In this case TxTpDataCnt specifies the offset of the first byte to be copied by the API call.
Parameters (inout):	None	
Parameters (out):	availableDataPtr	Indicates the remaining number of bytes that are available in the PduR Tx buffer. AvailableTxDataCntPtr can be used by TP modules that support dynamic payload lengths (e.g. Iso FrTp) to determine the size of the following CFs.
Return value:	BufReq_ReturnType	BUFREQ_OK: Data has been copied to the transmit buffer completely as requested. BUFREQ_E_BUSY: Request could not be fulfilled as the required amount of Tx data is not available. TP layer might retry later on. No data has been copied. BUFREQ_E_NOT_OK: Data has not been copied. Request failed.
Description:	This function is called by the transport protocol module to query the transmit data of an I-PDU segment. Each call to this function copies the next part of the transmit data until TpDataState indicates TP_DATARETRY. In this case the API restarts to copy the data beginning at the location indicated by TpTxDataCnt.	

⌋ ( )

8.6.4.8 <User>_SoAdTpTxConfirmation (PduR, CDD)

[SOAD181] ⌈

Service name:	<User_SoAdTpTxConfirmation>	

Syntax:	`void <User_SoAdTpTxConfirmation>(`
	`  PduIdType TxPduId,`
	`  NotifResultType result`
	`)`	

Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	TxPduId	Identification of the transmitted I-PDU.
	result	Result of the transmission of the I-PDU.

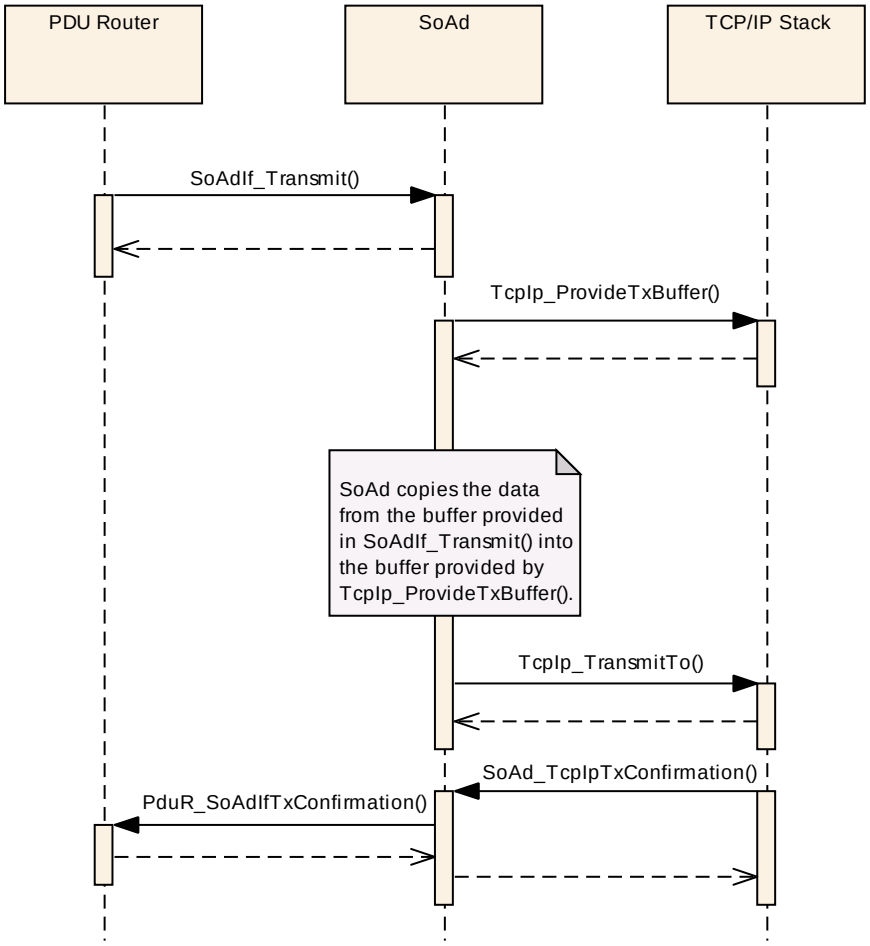
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	This function is called by a transport protocol module after the I-PDU has been transmitted on its network, the result will reveal if the transmission was successful or not.

⌋ ( )

9 Sequence diagrams and Transition Tables

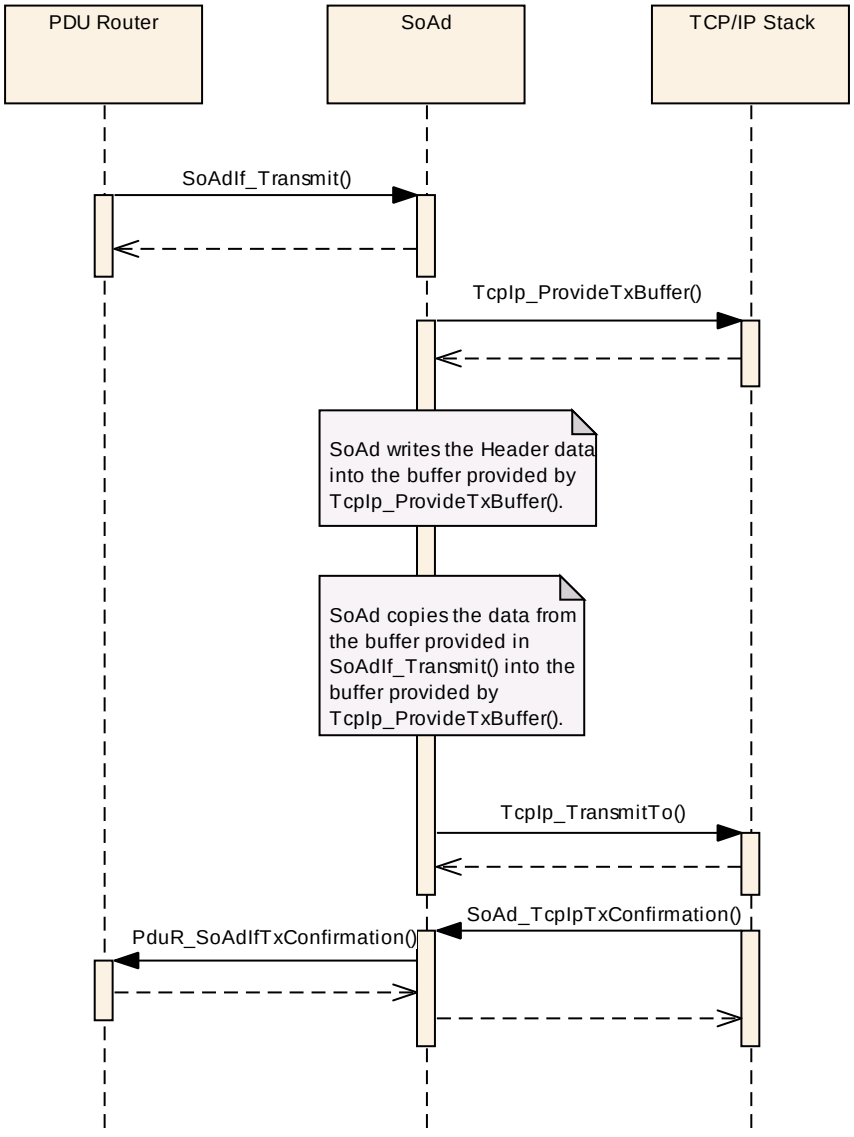
9.1 Transmission – IF type – AUTOSAR Call-Back – no Header

Call direction	*Action/Decision*	*Description*
PduR→SoAd	SoAdIf_Transmit()	
SoAd→TCP/IP	TcpIp_ProvideTxBuffer()	Allocate transmit buffer in the TCP/IP stack.
		SoAd copies the data from the buffer provided in SoAdIf_Transmit() into the buffer provided by the TCP/IP stack.
SoAd→TCP/IP	TcpIp_TransmitTo()	Have the TCP/IP stack send the data.
TCP/IP→SoAd	SoAd_TcpIpTxConfirmation()	TCP/IP stack confirms transmission up to SoAd and frees the occupied buffer.
SoAd→PduR	PduR_SoAdIfTxConfirmation()	SoAd confirms transmission up to PduR.



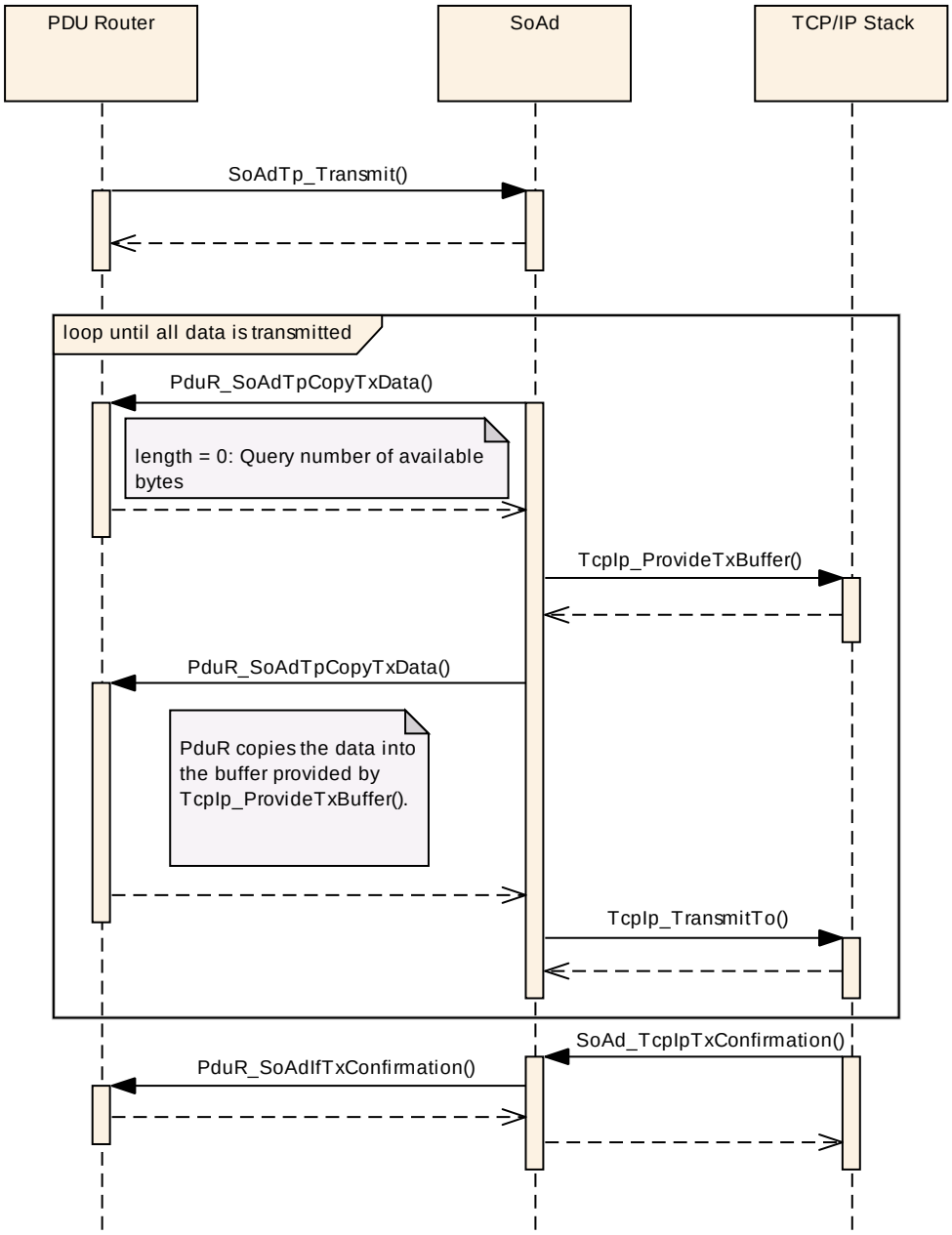
9.2 Transmission – IF type – AUTOSAR Call-Back – with Header

Call direction	*Action/Decision*	*Description*
PduR→SoAd	`SoAdIf_Transmit()`	
SoAd→TCP/IP	`TcpIp_ProvideTxBuffer()`	Allocate transmit buffer in the TCP/IP stack + Header bytes.
		SoAd writes the Header data into the buffer provided by the TCP/IP stack.
		SoAd copies the data from the buffer provided in `SoAdIf_Transmit()` into the buffer provided by the TCP/IP stack observing Header offset.
SoAd→TCP/IP	`TcpIp_TransmitTo()`	Have the TCP/IP stack send the data.
TCP/IP→SoAd	`SoAd_TcpIpTxConfirmation()`	TCP/IP stack confirms transmission up to SoAd.
SoAd→PduR	`PduR_SoAdIfTxConfirmation()`	SoAd confirms transmission up to PduR.



9.3 Transmission – TP type – AUTOSAR Call-Back – no Header

Call direction	*Action/Decision*	*Description*
PduR→SoAd	`SoAdTp_Transmit()`	
SoAd→PduR	`PduR_SoAdTpCopyTxData()`	How many bytes are available for transmission? If no more bytes are available, continue `SoAd_TcpIpTxConfirmation()`
SoAd→TCP/IP	`TcpIp_ProvideTxBuffer()`	Allocate transmit buffer in the TCP/IP stack
SoAd→PduR	`PduR_SoAdTpCopyTxData()`	Have the PduR copy the data into the TCP/IP stack. How many more bytes are available for transmission?
SoAd→TCP/IP	`TcpIp_TransmitTo()`	Have the TCP/IP stack send the data and free the occupied buffer.
TCP/IP→SoAd	`SoAd_TcpIpTxConfirmation()`	TCP/IP stack confirms transmission up to SoAd and frees allocated buffer.
		The SoAd may continue with step `TcpIp_ProvideTxBuffer()` without waiting for `SoAd_TcpIpTxConfirmation()`
SoAd→PduR	`PduR_SoAdTpTxConfirmation()`	SoAd confirms transmission up to PduR.

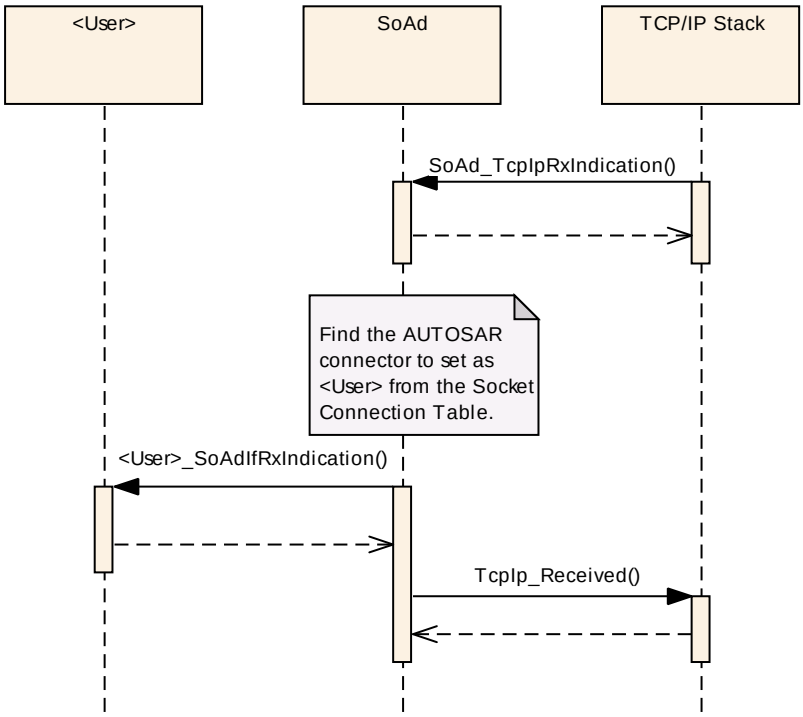

PDU Router
SoAd
TCP/IP Stack
SoAdTp_Transmit()
loop until all data is transmitted
PduR_SoAdTpCopyTxData()
length = 0: Query number of available bytes
TcpIp_ProvideTxBuffer()
PduR_SoAdTpCopyTxData()
PduR copies the data into the buffer provided by TcpIp_ProvideTxBuffer().
TcpIp_TransmitTo()
SoAd_TcpIpTxConfirmation()
PduR_SoAdIfTxConfirmation()

9.4 Transmission – TP type – AUTOSAR Call-Back – with Header

Call direction	*Action/Decision*	*Description*
PduR→SoAd	`SoAdTp_Transmit()`	
SoAd→PduR	`PduR_SoAdTpCopyTxData()`	How many bytes are available for transmission? If no more bytes are available, continue `SoAd_TcpIpTxConfirmation()`
SoAd→TCP/IP	`TcpIp_ProvideTxBuffer()`	Allocate transmit buffer in the TCP/IP stack + Header length
		SoAd writes the Header information into the buffer provided by the TCP/IP stack.
SoAd→PduR	`PduR_SoAdTpCopyTxData()`	Have the PduR copy the data into the TCP/IP stacks buffer observing header offset. How many bytes are available for transmission?
SoAd→TCP/IP	`TcpIp_TransmitTo()`	Have the TCP/IP stack send the data.
TCP/IP→SoAd	`SoAd_TcpIpTxConfirmation()`	TCP/IP stack confirms transmission up to SoAd.
		The SoAd may continue with step `TcpIp_ProvideTxBuffer()` without waiting for `SoAd_TcpIpTxConfirmation()`
SoAd→PduR	`PduR_SoAdTpTxConfirmation()`	SoAd confirms transmission up to PduR.

9.5 Reception – IF Type – AUTOSAR Call-Back – no Header

Call direction	*Action/Decision*	*Description*
TCP/IP→SoAd	SoAd_TcpIpRxIndication()	
	Find the AUTOSAR connector to set as <User> from the Socket Connection Table.	With the AUTOSAR call-back API enabled, the TCP/IP stack will not accept data if no valid entry in the Socket Connection Table exists.
SoAd→<User>	<User>_SoAdIfRxIndication()	Have the PduR copy the data from the TCP/IP stack.
SoAd→TCP/IP	TcpIp_Received()	Free the buffer in the TCP/IP stack.

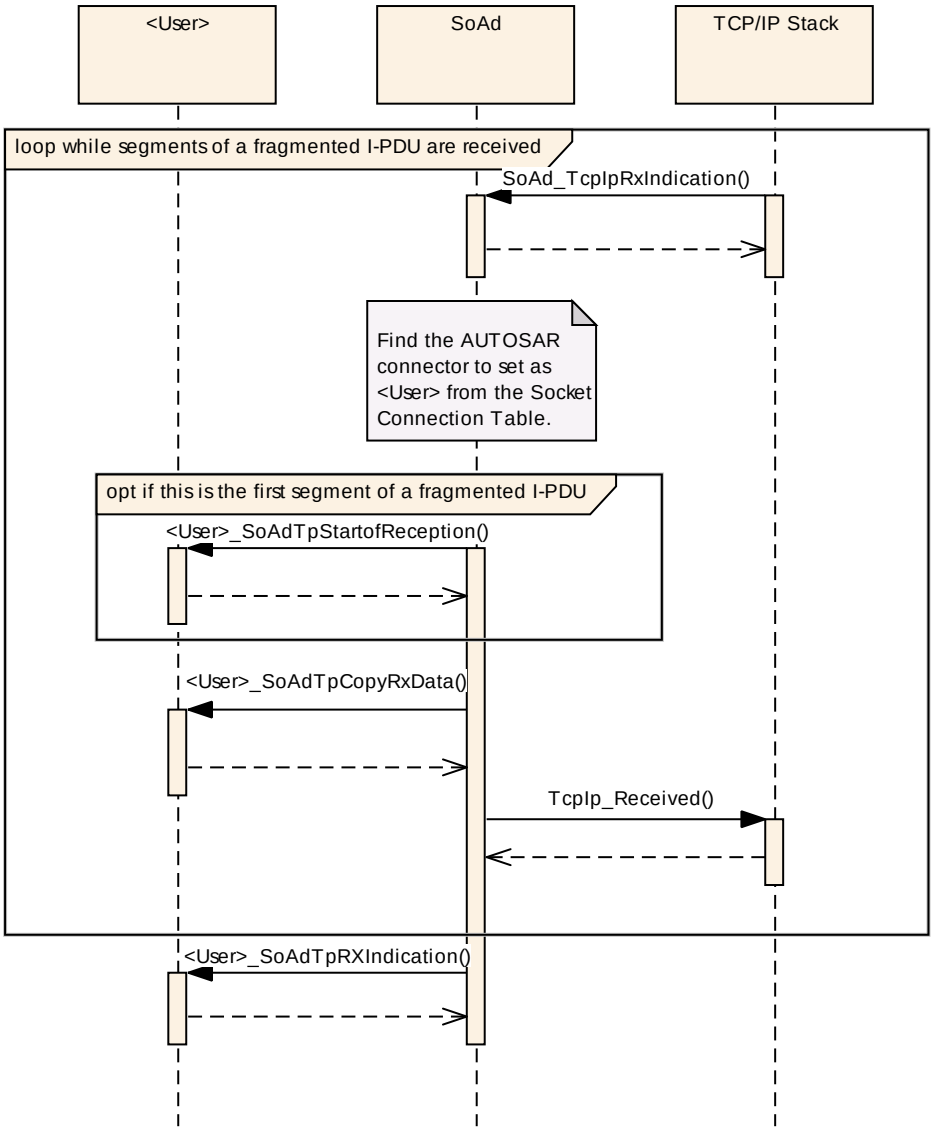


9.6 Reception – IF Type – AUTOSAR Call-Back – with Header

Call direction	*Action/Decision*	*Description*
TCP/IP→SoAd	SoAd_TcpIpRxIndication()	
	Find the Autosar connector to set as <User> from the Socket Connection Table.	With the AUTOSAR call-back API enabled, the TCP/IP stack will not accept data if no valid entry in the Socket Connection Table exists.
	Check the header information.	If Header info differs from Socket Connection Table, Report error SOAD_E_SDULENGTH to DEM and continue normal operation. Always use header info!
SoAd→<User>	<User>_SoAdIfRxIndication()	Have the PduR copy the data from the TCP/IP stack, adjust for header offset.
SoAd→TCP/IP	TcpIp_Received()	Free the buffer in the TCP/IP stack.

9.7 Reception – TP Type – AUTOSAR Call-Back – no Header

Call direction	*Action/Decision*	*Description*
TCP/IP→SoAd	SoAd_TcpIpRxIndication()	
	Find the Autosar connector to set as <User> from the Socket Connection Table.	With the AUTOSAR call-back API enabled, the TCP/IP stack will not accept data if no valid entry in the Socket Connection Table exists.
SoAd→<User>	If this is the first segment of a fragmented PDU <User>_SoAdTpStartofReception()	
SoAd→<User>	<User>_SoAdTpCopyRxData()	Have the PduR copy the data from the TCP/IP stack.
SoAd→TCP/IP	TcpIp_Received()	Free the buffer in the TCP/IP stack.
		when more data is received, continue at SoAd_TcpIpRxIndication()
SoAd→<User>	<User>_SoAdTpRXIndication()	


<User>
SoAd
TCP/IP Stack
loop while segments of a fragmented I-PDU are received
SoAd_TcpIpRxIndication()
Find the AUTOSAR connector to set as <User> from the Socket Connection Table.
opt if this is the first segment of a fragmented I-PDU
<User>_SoAdTpStartofReception()
<User>_SoAdTpCopyRxData()
TcpIp_Received()
<User>_SoAdTpRXIndication()

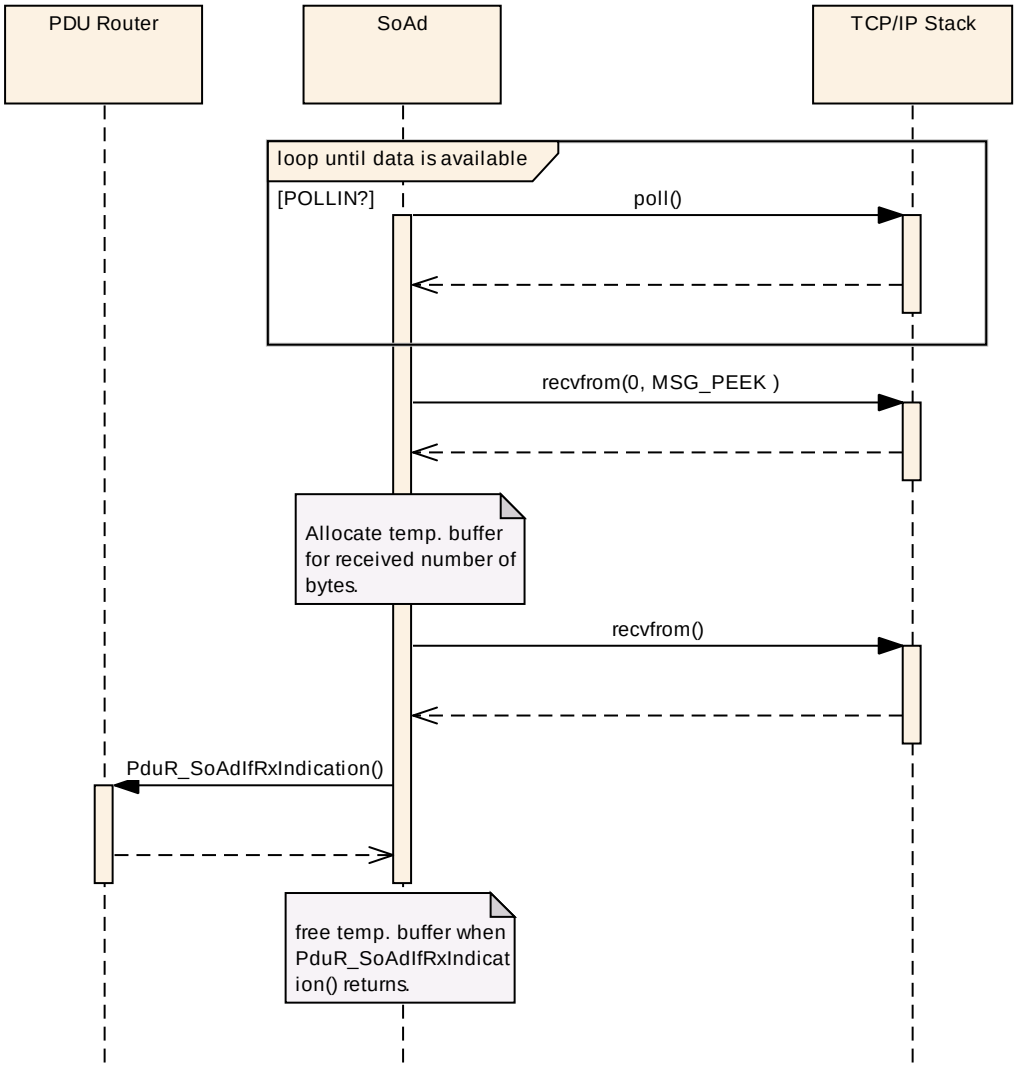
9.8 Reception – TP Type – AUTOSAR Call-Back – with Header

Call direction	*Action/Decision*	*Description*
TCP/IP→SoAd	SoAd_TcpIpRxIndication()	
	Find the Autosar connector to set as <User> from the Socket Connection Table.	With the AUTOSAR call-back API enabled, the TCP/IP stack will not accept data if no valid entry in the Socket Connection Table exists.
SoAd→<User>	If this is the first segment of a fragmented PDU <User>_SoAdTpStartofReception()	
SoAd→<User>	<User>_SoAdTpCopyRxData()	Have the PduR copy the data from the TCP/IP stack. If this is the first segment of a fragmented PDU adjust pointer to exclude Header.
SoAd→TCP/IP	TcpIp_Received()	Free the buffer in the TCP/IP stack.
		When more data is received, continue at SoAd_TcpIpRxIndication()
SoAd→<User>	<User>_SoAdTpRXIndication()	

9.9 Reception – IF Type – BSD Sockets – no Header – UDP

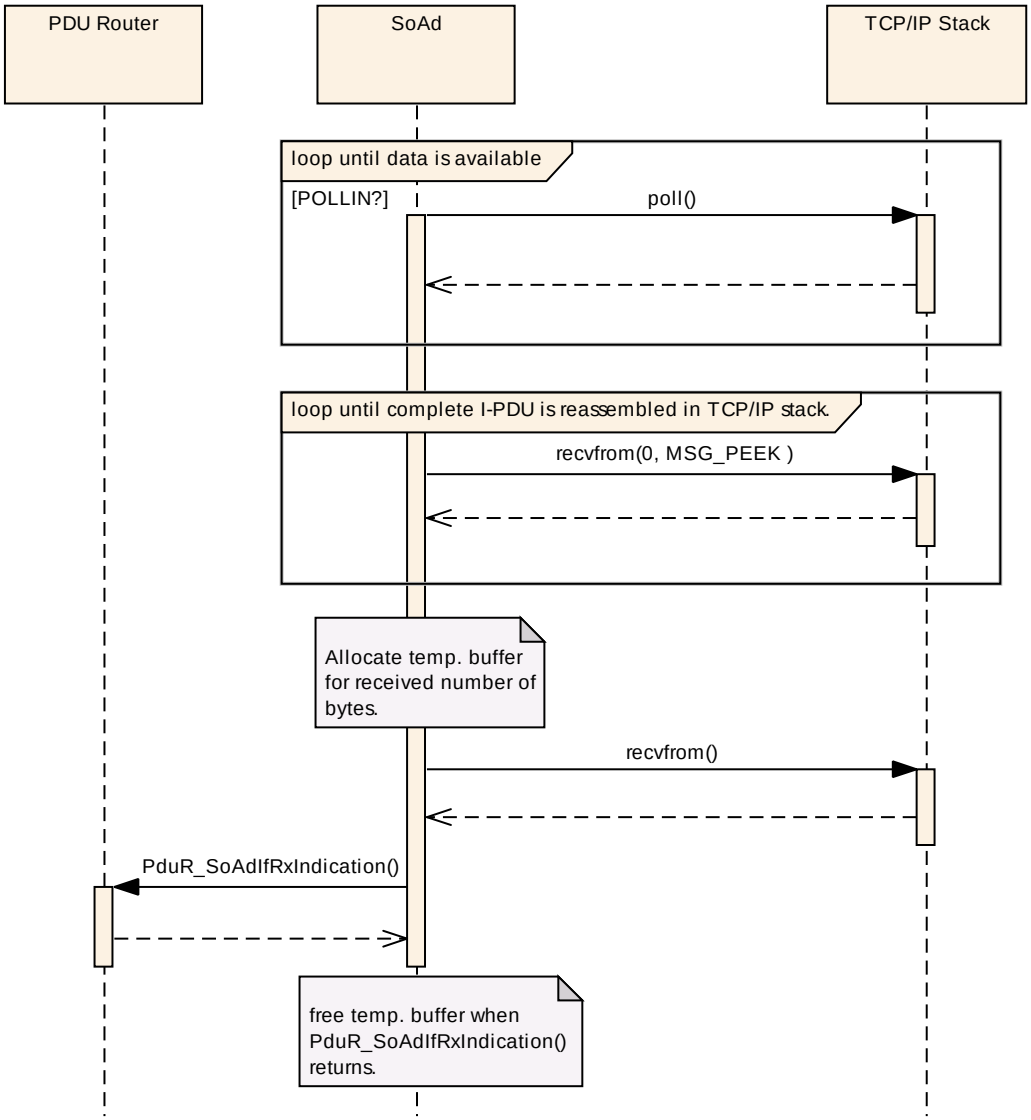
Note: `recvfrom()`: For message-based sockets, such as `SOCK_DGRAM`, the entire message shall be read in a single operation. If a message is too long to fit in the supplied buffer, and `MSG_PEEK` is not set in the `flags` argument, the excess bytes shall be discarded.

Call direction	*Action/Decision*	*Description*
SoAd→TCP/IP	`poll()`	check for available data
	`POLLIN`?	NO: continue `poll()`
	Valid entry in Socket Connection Table?	NO: Terminate this Transistion Table, continue in Chapter 9.17
SoAd→TCP/IP	`recvfrom()` with `MSG_PEEK` and zero buffer length	keep all data in TCP/IP stack
	Is `SduLength` in Socket Connection Table == length of packet?	NO: Report error `SOAD_E_SDULENGTH` to DEM.
	Is complete SDU (from length given in header) contained in the packet (from number of bytes received)	NO: Report error `SOAD_E_SDULENGTH` to DEM.
	Allocate temp. buffer for received number of bytes	No buffer available: continue `poll()` Report error `SOAD_E_UPPERBUFF` to DEM
SoAd→TCP/IP	`recvfrom()`	Copy data from TCP/IP stack to SoAd buffer
SoAd→PduR	`PduR_SoAdIfRxIndication()`	Pass buffer pointer up the AUTOSAR stack
	free temp. buffer when `RxIndication()` returns	


PDU Router
SoAd
TCP/IP Stack
loop until data is available
[POLLIN?]
poll()
recvfrom(0, MSG_PEEK)
Allocate temp. buffer for received number of bytes.
recvfrom()
PduR_SoAdIfRxIndication()
free temp. buffer when PduR_SoAdIfRxIndication() returns.

9.10 Reception – IF Type – BSD Sockets – no Header – TCP

Call direction	*Action/Decision*	*Description*
SoAd→TCP/IP	`poll()`	check for available data
	`POLLIN`?	NO: continue `poll()`
	Valid entry in Socket Connection Table?	NO: Terminate this Transistion Table, continue in Chapter 9.17
SoAd→TCP/IP	`recvfrom()` with `MSG_PEEK` and zero buffer length	keep all data in TCP/IP stack
	Is complete SDU (from length given in Socket Connection Table) contained in the packet (from number of bytes received)	NO: continue `poll()` – wait for complete SDU to be reassembled in TCP/IP stack.
	Allocate temp. buffer for SDU length in socket connection Table.	No buffer available: continue `poll()` Report error `SOAD_E_UPPERBUFF` to DEM
SoAd→TCP/IP	`recvfrom()` with SDU length.	Packet might contain multiple instance of the SDU. Copy data of first instance from TCP/IP stack to SoAd buffer
SoAd→PduR	`PduR_SoAdIfRxIndication()`	Pass buffer pointer up the AUTOSAR stack
	free temp. buffer when `RxIndication()` returns	


PDU Router
SoAd
TCP/IP Stack
loop until data is available
[POLLIN?]
poll()
loop until complete I-PDU is reassembled in TCP/IP stack.
recvfrom(0, MSG_PEEK)
Allocate temp. buffer for received number of bytes.
recvfrom()
PduR_SoAdIfRxIndication()
free temp. buffer when PduR_SoAdIfRxIndication() returns.

9.11 Reception – IF Type – BSD Sockets – with Header – UDP

Note: `recvfrom()`: For message-based sockets, such as `SOCK_DGRAM`, the entire message shall be read in a single operation. If a message is too long to fit in the supplied buffer, and `MSG_PEEK` is not set in the `flags` argument, the excess bytes shall be discarded.

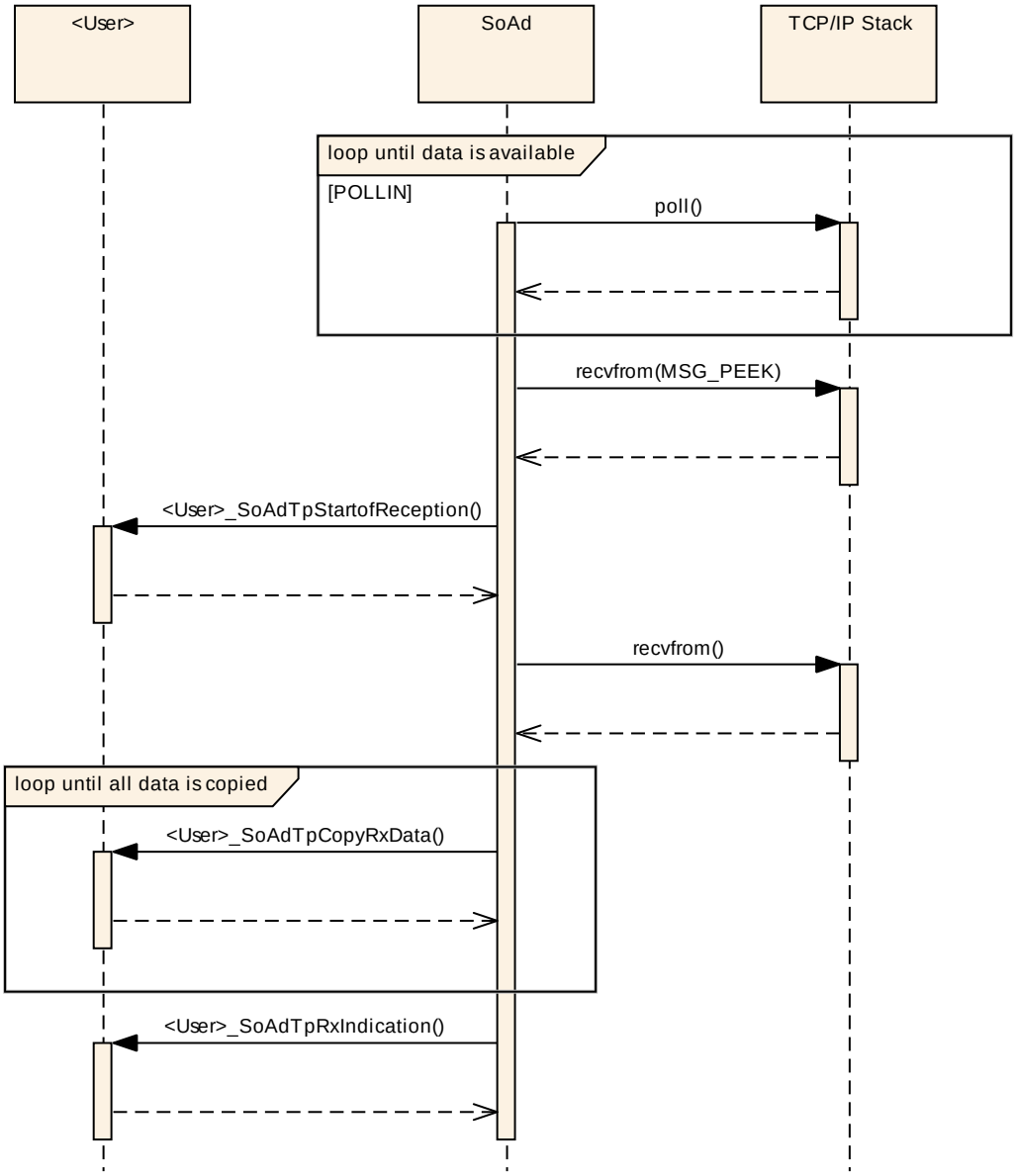
Call direction	*Action/Decision*	*Description*
SoAd→TCP/IP	`poll()`	check for available data
	`POLLIN`?	NO: continue `poll()`
	At least one valid entry in Socket Connection Table?	NO: Terminate this Transistion Table, continue in Chapter 9.17
		Allocate temp. buffer `sizeof(Header)`
SoAd→TCP/IP	`recvfrom()` with `MSG_PEEK`	read header, keep all data in TCP/IP stack
	Is `SduLength` in Socket Connection Table == length in Header?	NO: Report error `SOAD_E_SDULENGTH` to DEM.
	Is complete SDU (from length given in header) contained in the packet (from number of bytes received)	NO: Report error `SOAD_E_SDULENGTH` to DEM. Continue with number of bytes received instead of header information.
	Allocate temp. buffer for SDU length in header + header length.	No buffer available: continue `poll()` Report error `SOAD_E_UPPERBUFF` to DEM
SoAd→TCP/IP	`recvfrom()` with SDU length	Copy data from TCP/IP stack to SoAd buffer
SoAd→PduR	`PduR_SoAdIfRxIndication()`	Pass buffer pointer up the AUTOSAR stack, adjust for header length
	free temp. buffer when `RxIndication` returns	

9.12 Reception – IF Type – BSD Sockets – with Header – TCP

Call direction	*Action/Decision*	*Description*
SoAd→TCP/IP	poll()	check for available data
	POLLIN?	NO: continue poll()
	Valid entry in Socket Connection Table?	NO: Terminate this Transistion Table, continue in Chapter 9.17
		Allocate temp. buffer sizeof(Header)
SoAd→TCP/IP	recvfrom() with MSG_PEEK	read header, keep all data in TCP/IP stack
	Is SduLength in Socket Connection Table == length in Header?	NO: Report error SOAD_E_SDULENGTH to DEM.
	Is complete SDU (from length given in header) contained in the received stream (from number of bytes received)	NO: continue poll() – wait for complete SDU to be reassembled in TCP/IP stack.
	Allocate temp. buffer for SDU length in header	Stream might contain multiple SDUs No buffer available: continue poll() Report error SOAD_E_UPPERBUFF to DEM
SoAd→TCP/IP	recvfrom() with SDU length	Copy data from TCP/IP stack to SoAd buffer
SoAd→PduR	PduR_SoAdIfRxIndication()	Pass buffer pointer up the AUTOSAR stack, adjust for header length
	free temp. buffer when RxIndication returns	

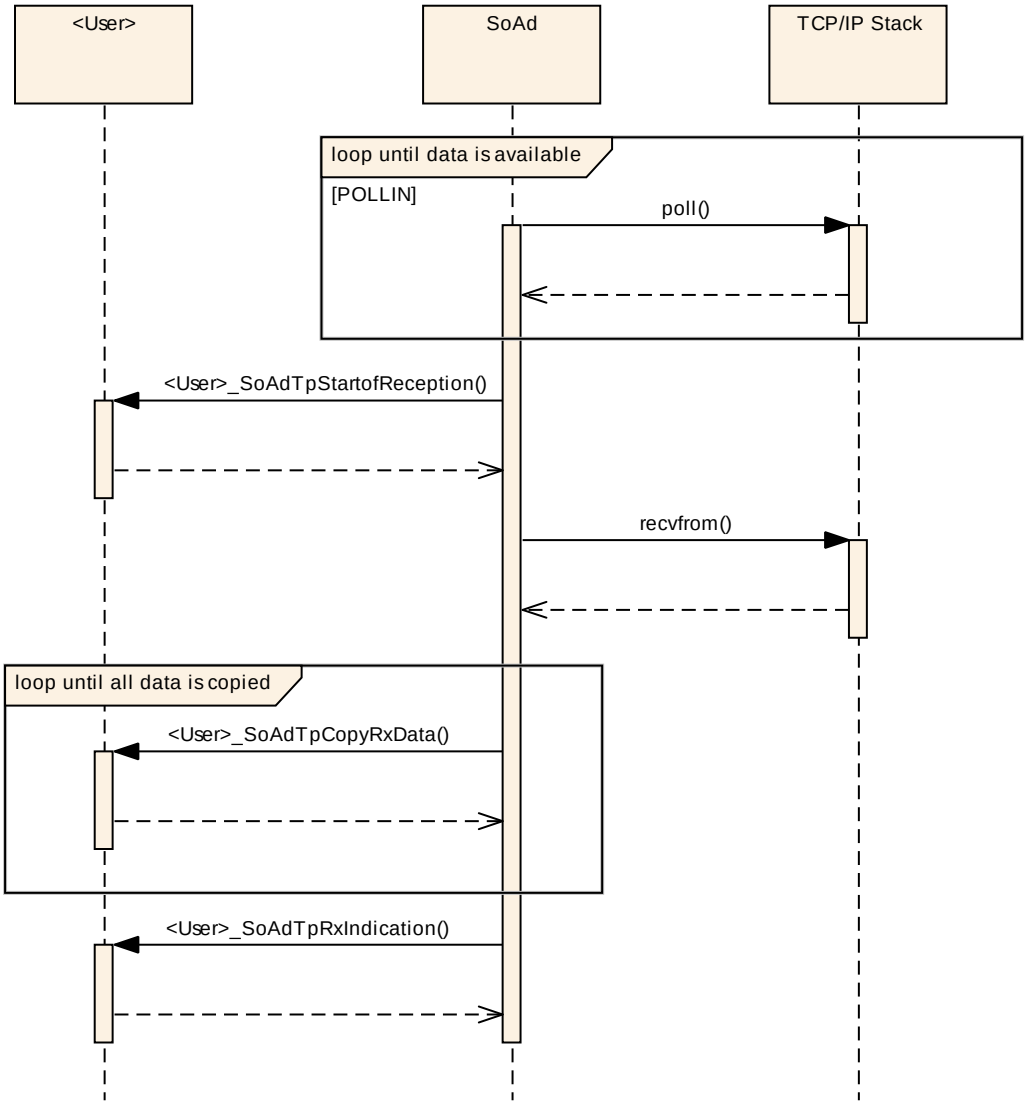
9.13 Reception – TP Type – BSD Sockets – with Header – UDP

Call direction	*Action/Decision*	*Description*
SoAd→TCP/IP	`poll()`	check for available data
	`POLLIN`?	NO: continue `poll()`
	Valid entry in Socket Connection Table?	NO: Terminate this Transistion Table, continue in Chapter 9.17
	Allocate temp. buffer `sizeof(Header)`	
SoAd→TCP/IP	`recvfrom()` with `MSG_PEEK`	read header, keep all data in TCP/IP stack
	Is `SduLength` in Socket Connection Table == length in Header?	NO: Report error `SOAD_E_SDULENGTH` to DEM.
SoAd→<User>	`<User>_SoAdTpStartofReception()`	
	Allocate temp. buffer for received packet	
SoAd→TCP/IP	`recvfrom()`	
SoAd→<User>	`<User>_SoAdTpCopyRxData()`	adjust for Header in first copy operation
	continue `<User>_SoAdTpCopyRxData()` until all data is copied to <User>	
SoAd→<User>	`<User>_SoAdTpRxIndication()`	


<User>
SoAd
TCP/IP Stack
loop until data is available
[POLLIN]
poll()
recvfrom(MSG_PEEK)
<User>_SoAdTpStartofReception()
recvfrom()
loop until all data is copied
<User>_SoAdTpCopyRxData()
<User>_SoAdTpRxIndication()

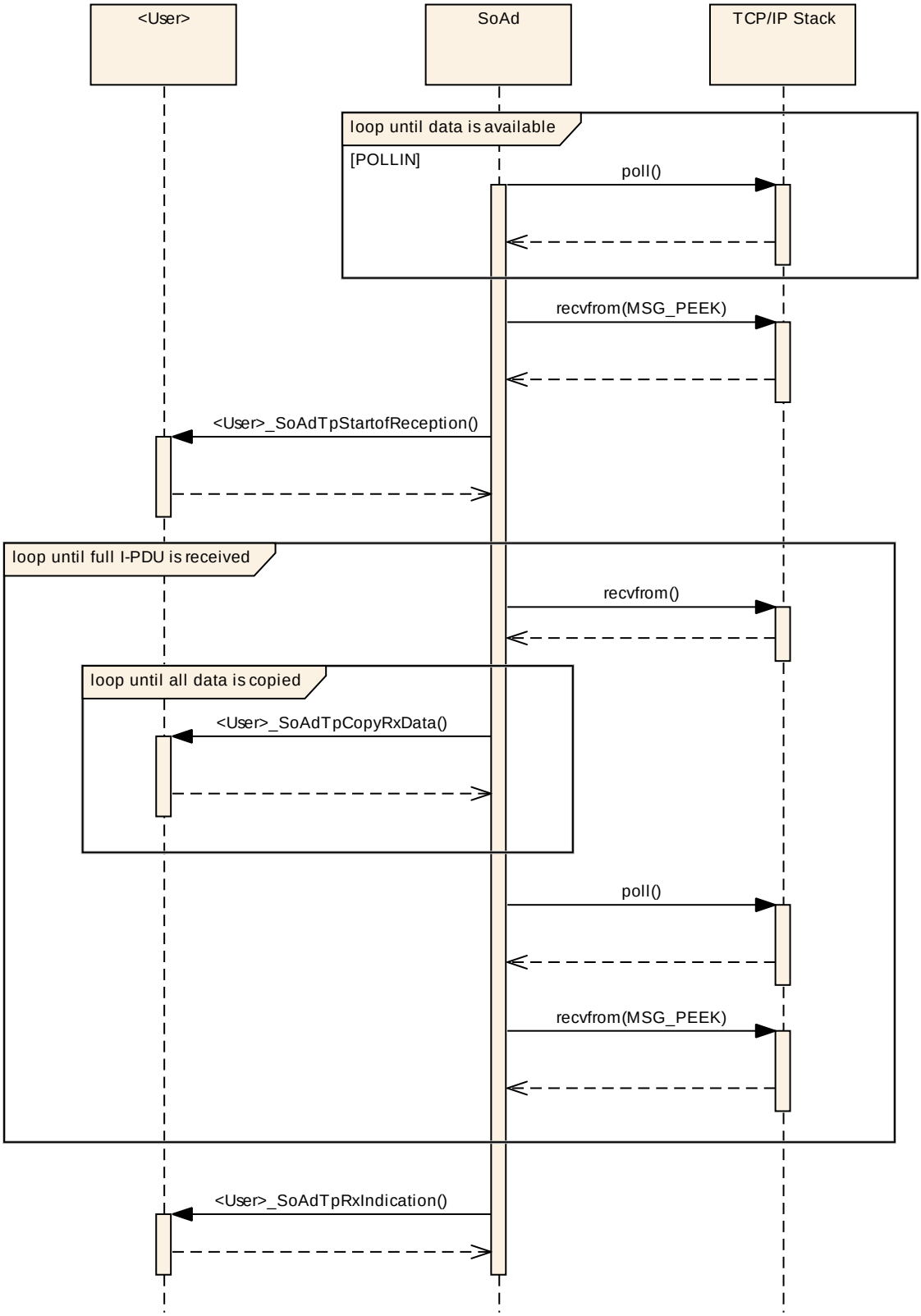
9.14 Reception – TP Type – BSD Sockets – no Header – UDP

Call direction	*Action/Decision*	*Description*
SoAd→TCP/IP	`poll()`	check for available data
	`POLLIN`?	NO: continue `poll()`
	Valid entry in Socket Connection Table?	NO: Terminate this Transistion Table, continue in Chapter 9.17
	Is `SduLength` in Socket Connection Table == length of the packet?	NO: Report error `SOAD_E_SDULENGTH` to DEM.
SoAd→<User>	`<User>_SoAdTpStartofReception()`	
	Allocate temp. buffer for received packet	
SoAd→TCP/IP	`recvfrom()`	
SoAd→<User>	`<User>_SoAdTpCopyRxData()`	
	continue `<User>_SoAdTpCopyRxData()` until all data is copied to <User>	
SoAd→<User>	`<User>_SoAdTpRxIndication()`	


<User>
SoAd
TCP/IP Stack
loop until data is available
[POLLIN]
poll()
<User>_SoAdTpStartofReception()
recvfrom()
loop until all data is copied
<User>_SoAdTpCopyRxData()
<User>_SoAdTpRxIndication()

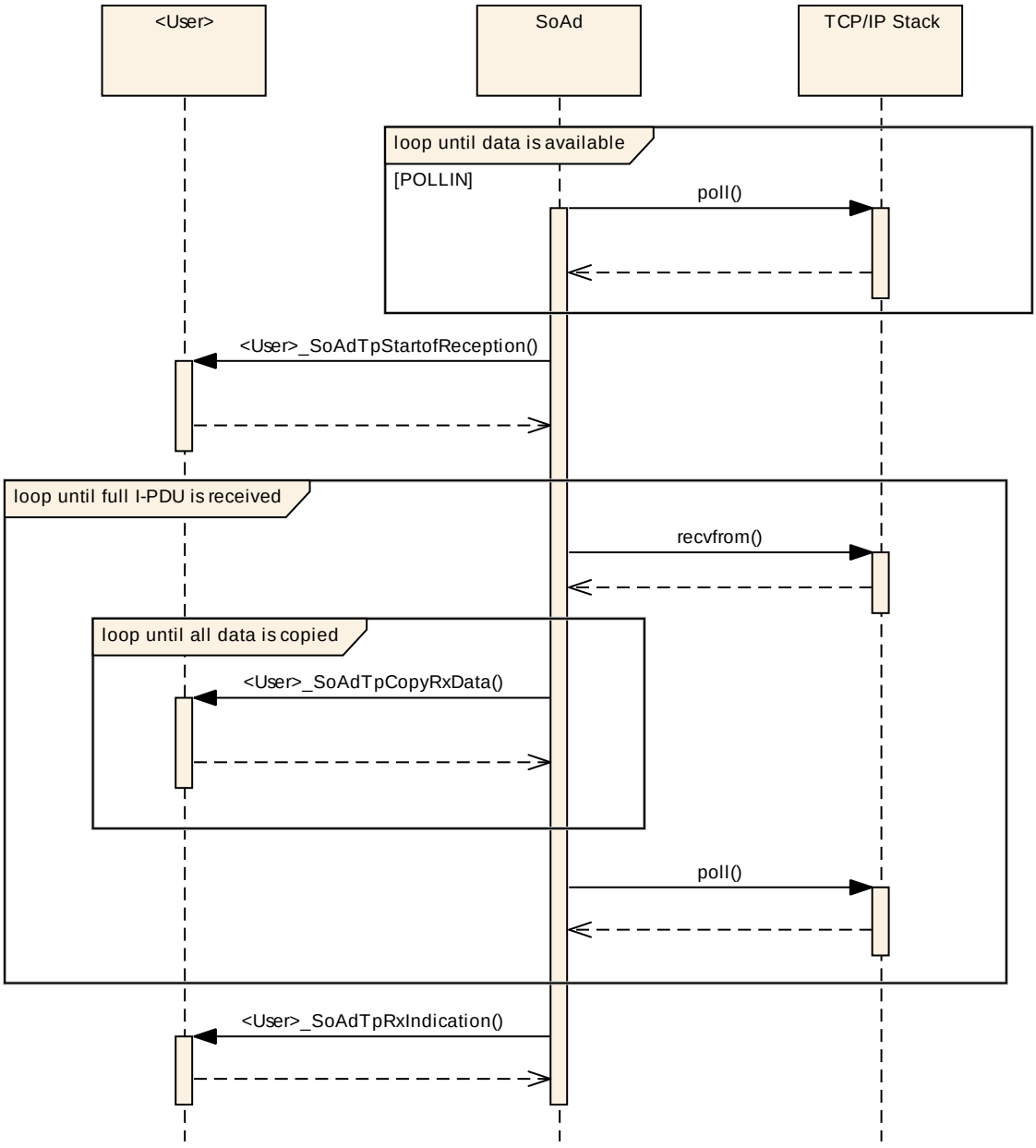
9.15 Reception – TP Type – BSD Sockets – with Header – TCP

Call direction	*Action/Decision*	*Description*
SoAd→TCP/IP	`poll()`	check for available data
	`POLLIN`?	NO: continue `poll()`
	Valid entry in Socket Connection Table?	NO: Terminate this Transistion Table, continue in Chapter 9.17
	Allocate temp. buffer `sizeof(Header)`	
SoAd→TCP/IP	`recvfrom()` with `MSG_PEEK`	read header, keep all data in TCP/IP stack
	Is `SduLength` in Socket Connection Table == length in Header?	NO: Report error `SOAD_E_SDULENGTH` to DEM.
SoAd→<User>	`<User>_SoAdTpStartofReception()`	
	Allocate temp. buffer for received packet	
SoAd→TCP/IP	`recvfrom()`	Prevent reading header or parts of next PDU in the same TCP stream!
SoAd→<User>	`<User>_SoAdTpCopyRxData()`	
	continue `<User>_SoAdTpCopyRxData()` until all data is copied to <User>	
	free temp. buffer	
SoAd→TCP/IP	`poll()`	check for available data
	`POLLIN`?	NO: continue `poll()`
SoAd→TCP/IP	`recvfrom()` with `MSG_PEEK`	copy no data, just determin length of received data.
	Allocate temp. buffer for received packet	
	continue `recvfrom()` until full PDU is received	
SoAd→<User>	`<User>_SoAdTpRxIndication()`	


<User>
SoAd
TCP/IP Stack
loop until data is available
[POLLIN]
poll()
recvfrom(MSG_PEEK)
<User>_SoAdTpStartofReception()
loop until full I-PDU is received
recvfrom()
loop until all data is copied
<User>_SoAdTpCopyRxData()
poll()
recvfrom(MSG_PEEK)
<User>_SoAdTpRxIndication()

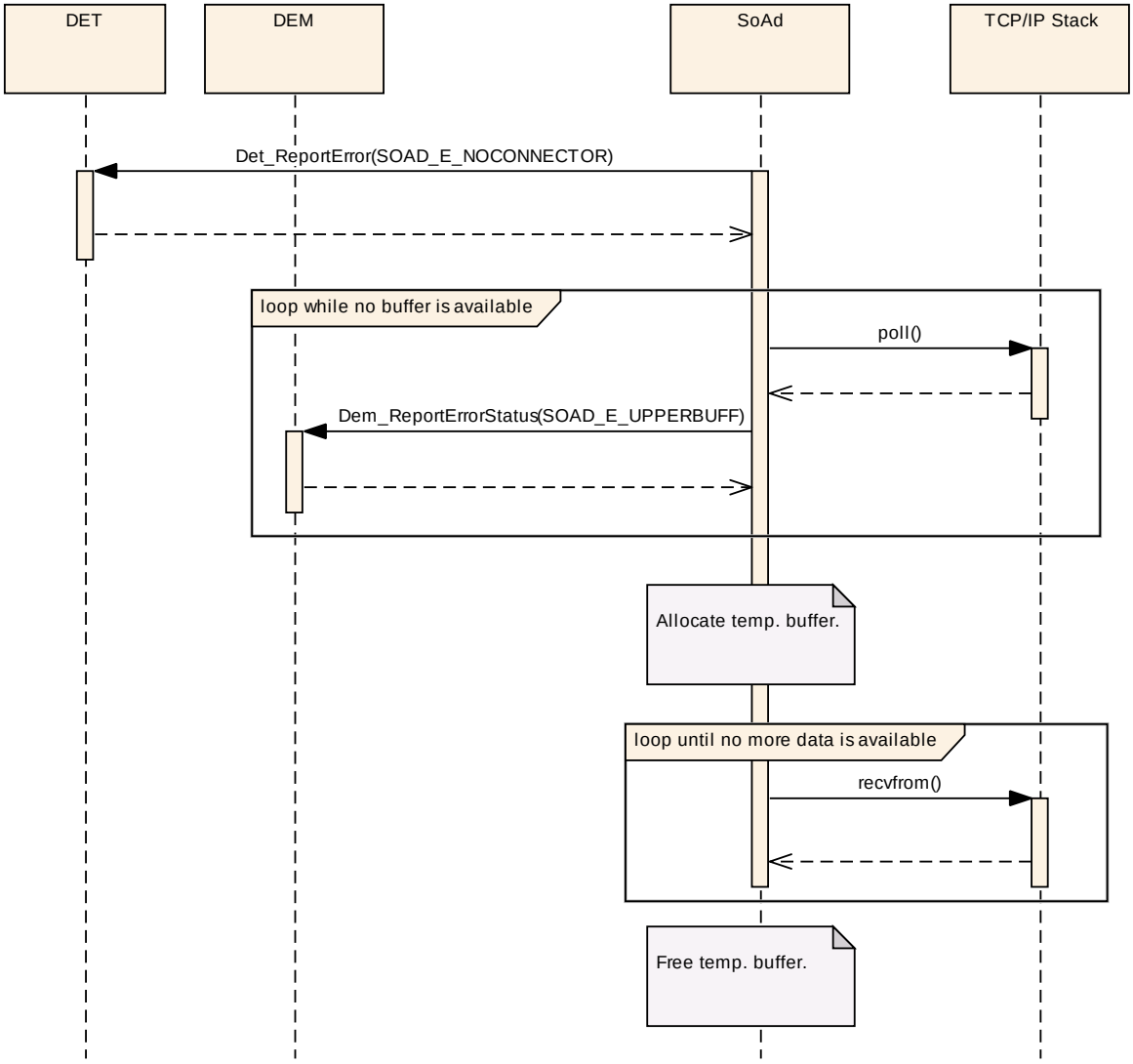
9.16 Reception – TP Type – BSD Sockets – no Header – TCP

Call direction	*Action/Decision*	*Description*
SoAd→TCP/IP	`poll()`	check for available data
	`POLLIN`?	NO: continue `poll()`
	Valid entry in Socket Connection Table?	NO: Terminate this Transistion Table, continue in Chapter 9.17
SoAd→<User>	`<User>_SoAdTpStartofReception()`	
	Allocate temp. buffer for received packet	
SoAd→TCP/IP	`recvfrom()`	Prevent reading parts of next PDU in the same TCP stream! Use length in Socket Connection Table.
SoAd→<User>	`<User>_SoAdTpCopyRxData()`	
	continue `<User>_SoAdTpCopyRxData()` until all data is copied to <User>	
SoAd→TCP/IP	`poll()`	check for available data
	`POLLIN`?	NO: continue `poll()`
	continue `recvfrom()` until full PDU is received	
SoAd→<User>	`<User>_SoAdTpRxIndication()`	


<User>
SoAd
TCP/IP Stack
loop until data is available
[POLLIN]
poll()
<User>_SoAdTpStartofReception()
loop until full I-PDU is received
recvfrom()
loop until all data is copied
<User>_SoAdTpCopyRxData()
poll()
<User>_SoAdTpRxIndication()

9.17 Reception Error Handling – BSD Sockets

Call direction	*Action/Decision*	*Description*
SoAd→DET		Report error SOAD_E_NOCONNECTOR to DET.
	Allocate temp. buffer of any size.	No buffer available: continue `poll()` Report error SOAD_E_UPPERBUFF to DEM
SoAd→TCP/IP	`recvfrom()`	Copy data from TCP/IP stack to SoAd buffer Repeat until all data is read.
		Free temp. buffer.



10 Configuration specification

In general, this chapter defines configuration parameters and their clustering into containers. In order to support the specification Chapter 10.1 describes fundamentals. It also specifies a template (table) you shall use for the parameter specification. We intend to leave Chapter 10.1 in the specification to guarantee comprehension.

Chapter 10.2 specifies the structure (containers) and the parameters of module SoAd.

Chapter 0 specifies published information of module SoAd.

10.1 How to read this chapter

In addition to this section, it is highly recommended to read the documents:

- AUTOSAR Layered Software Architecture [1].
- AUTOSAR ECU Configuration Specification [6].
  This document describes the AUTOSAR configuration methodology and the AUTOSAR configuration metamodel in detail.

The following is only a short survey of the topic and it will not replace the ECU Configuration Specification document.

10.1.1 Configuration and configuration parameters

Configuration parameters define the variability of the generic part(s) of an implementation of a module. This means that only generic or configurable module implementation can be adapted to the environment (software/hardware) in use during system and/or ECU configuration.

The configuration of parameters can be achieved at different times during the software process: before compile time, before link time or after build time. In the following, the term “configuration class” (of a parameter) shall be used in order to refer to a specific configuration point in time.

10.1.2 Variants

Variants describe sets of configuration parameters. E.g., variant 1: only pre-compile time configuration parameters; variant 2: mix of pre-compile- and post build time configuration parameters. In one variant a parameter can only be of one configuration class.

10.1.3 Containers

Containers structure the set of configuration parameters. This means:

- all configuration parameters are kept in containers.
- (sub-) containers can reference (sub-) containers. It is possible to assign a multiplicity to these references. The multiplicity then defines the possible number of instances of the contained parameters.
- the configuration parameters logically belong together (e.g., general parameters which are valid for the entire module NVRAM manager)
- the configuration parameters need to be instantiated (e.g., parameters of the memory block specification of the NVRAM manager – those parameters must be instantiated for each memory block)>

10.1.4 Specification template for configuration parameters

The following tables consist of three sections:

- the general section
- the configuration parameter section
- the section of included/referenced containers

SWS Item	SOADxxx
Container Name	<Identifies the container by a name, e.g., CanDriverConfiguration>
Description	<Explains the intention and the content of the container .>
Configuration Parameters	

Name	<Identifies the parameter by name. The naming convention shall follow BSW00408.>		
Description	<Explains the intention of the configuration parameter.>		
Type	<Specify the type of the parameter (e.g., uint8..uint32) if possible or mark it "--">		
Unit	<Specify the unit of the parameter (e.g., ms) if possible or mark it "--" >		
Range	<Specify the range (or possible values) of the parameter (e.g., 1..15, ON,OFF) if possible or mark it "--">	<Describe the value(s) or ranges.>	
Configuration Class	Pre-compile	see[1]	<Refer here to (a) variant(s).>
	Link time	see[2]	<Refer here to (a) variant(s).>
	Post Build	see[3]	<Refer here to (a) variant(s).>
Scope	<Describe the scope of the parameter if known or mark it as "- -". The scope describes the impact of the configuration parameter: Does the setting affect only one instance of the module (instance), all instances of this module (module), the ECU or a network. Possible values of scope : instance, module, ECU, network>		
Dependency	<Describe the dependencies with respect to the scope if known ot mark it as "- -".>		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
<Reference a valid (sub)container by its name, e.g.,CanController>	<Specifies the possible number of instances of the referenced container and its contained configuration parameters. Possible values: <multiplicity> <min_multiplicity..max_multiplicity> >	<Describe the scope of the referenced sub-container if known or mark it as "- -". The scope describes the impact of the configuration parameter: Does the setting affect only one instance of the module (instance), all instances of this module (module), the ECU or a network. Possible values of scope : instance, module, ECU, network> <Describe the dependencies with respect to the scope if known ot mark it as "- -".>

Pre-compile time
specifies whether the configuration parameter shall be of configuration class Pre-compile time or not

Label	***Description***
X	The configuration parameter shall be of configuration class Pre-compile time.
--	The configuration parameter shall never be of configuration class Pre-compile time.

Link time
specifies whether the configuration parameter shall be of configuration class Link time or not

[1] see the explanation below this table - Pre-compile time
[2] see the explanation below this table - Link time
[3] see the explanation below this table - Post Build

Label	Description
X	The configuration parameter shall be of configuration class Link time.
--	The configuration parameter shall never be of configuration class Link time.

Post Build

specifies whether the configuration parameter shall be of configuration class Post Build or not

Label	Description
X	The configuration parameter shall be of configuration class Post Build and no specific implementation is required.
L	Loadable - the configuration parameter shall be of configuration class Post Build and only one configuration parameter set resides in the ECU.
M	Multiple – the configuration parameter shall be of configuration class Post Build and is selected out of a set of multiple parameters by passing a dedicated pointer to the init function of the module.
--	The configuration parameter shall never be of configuration class Post Build.

10.2 Containers and configuration parameters

The configuration parameters as defined in this chapter are used to create a data model for an AUTOSAR tool chain. The realization in the code is implementation specific.
The configuration parameters as defined in this chapter are used to create a data model for an AUTOSAR tool chain. The realization in the code is implementation specific.
The configuration parameters are divided into parameters used to enable features, parameters affecting all instances of the UdpNm and parameters affecting the respective instances of the UdpNm.

[SOAD001] ⌈All configuration items shall be located outside the kernel of the module.⌋( )

[SOAD208] ⌈All timing parameters given as EcucFloatParamDef in unit seconds in the configuration, shall not necessarily be implemented as FLOAT, but their type and unit shall be adapted to the values required.⌋( )

10.2.1 Variants

Variant 1: All configuration parameters shall be configurable at pre-compile time.
Use case: Source code optimization.

Variant 2: All configuration parameters of the container `SoAd_GlobalConfig` related to enable or disable an optional feature shall be configurable at pre-compile time; the remaining configuration parameters shall be configurable at link time.

Use case: Object code.

Variant 3: The parameters contained in SoAd_ChannelConfig are configurable at post-build time. The parameters contained in SoAd_GlobalConfig are configurable at pre-compile time
Use case: ECU configuration can be flashed (L) and selected during initialization phase (M).

Note: The possibility to select a configuration (post-build time type L) is explicitly mentioned for Variant 3 only, but from a technical perspective it is also possible to provide this configuration variant for variant 1 and 2.

10.2.2 SoAd

SWS Item	**SOAD001_Conf :**
Module Name	*SoAd*
Module Description	Configuration of the SoAd (Socket Adaptor) module.

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
SoAdDemEventParameterRefs	0..1	Container for the references to DemEventParameter elements which shall be invoked using the API Dem_ReportErrorStatus API in case the corresponding error occurs. The EventId is taken from the referenced DemEventParameter's DemEventId value. The standardized errors are provided in the container and can be extended by vendor specific error references.
SoAdDoIpConfig	1	This container contains all global configuration parameters of the DoIP plug-in.
SoAdDoIpRoute	1	A SoAd_DoIP_Route allocates a PDU ID to a combination of a DoIP source and a DoIP target address.
SoAdGeneral	1	This container contains all global configuration parameters of SoAd configured from the Pdu Router Module perspective.
SoAdPduRoute	1..*	Describes the path of a PDU from the PDU Router to the socket in the TCP/IP stack for transmission.
SoAdSocketConnection	1..*	Information required to receive and transmit data via the TCP/IP stack on a particular connection.
SoAdSocketRoute	1..*	Describes the path of a PDU from a socket in the TCP/IP stack to the PDU Router after reception in the TCP/IP Stack.

[SOAD002] ⌈The Global Scope specifies configuration parameter that shall be defined in the module's configuration header file `SoAd_Cfg.h`.⌋ ( )

10.2.3 SoAdGeneral

SWS Item	**SOAD003_Conf :**
Container Name	SoAdGeneral{SoAd_Global_Config}
Description	This container contains all global configuration parameters of SoAd configured from the Pdu Router Module perspective.
Configuration Parameters	

SWS Item	**SOAD061_Conf :**		
Name	SoAdBufferMemorySize {SOAD_BUFFER_MEMORY_SIZE}		
Description	Memory size reserved for SoAd buffers.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	

Scope / Dependency	

SWS Item	**SOAD013_Conf :**		
Name	SoAdCallbackApi {SOAD_CALL_BACK_API}		
Description	True if the TCP/IP stack supports the AUTOSAR Call-back API in addition to the Berkeley Socket API. TRUE: TCP/IP Stack supports AUTOSAR callback API FALSE: TCP/IP Stack supports only BSD Sockets.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD002_Conf :**		
Name	SoAdDevErrorDetect {SOAD_DEV_ERROR_DETECT}		
Description	Pre-processor switch for enabling development error detection support.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD011_Conf :**		
Name	SoAdDoIpActive {SOAD_DOIP_ACTIVE}		
Description	True if a DoIP protocol plug-in is available.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD005_Conf :**		
Name	SoAdDoIpVersionInfoApi {DOIP_VERSION_INFO_API}		
Description	Switches the DoIP_GetVersionInfo() API ON or OFF.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	

Scope / Dependency	

SWS Item	**SOAD039_Conf :**		
Name	SoAdIPv6AddressEnabled {SOAD_IPv6_ADDRESS_ENABLED}		
Description	Allows for increased memory allocation to store IPv6 addresses. TRUE: Enables support for IPv6 addresses FALSE: Only IPv4 addresses are supported		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD062_Conf :**		
Name	SoAdMainFunctionPeriod {SOAD_MainFunction_Period}		
Description	Determines the frequency at which the SoAd_MainFunction() is called in [s].		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	`0 .. INF`		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD015_Conf :**		
Name	SoAdMaxOpenSockets {SOAD_MAX_OPEN_SOCKETS}		
Description	Specifies the number of sockets that will be open at any one time.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD014_Conf :**		
Name	SoAdPollingInterval {SOAD_POLLING_INTERVAL}		
Description	Specifies the interval at which the SoAd shall poll the TCP/IP stack for new information in [s].		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	`0 .. INF`		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	

Scope / Dependency	

SWS Item	**SOAD012_Conf :**		
Name	SoAdSocketCount {SOAD_SOCKET_COUNT}		
Description	Number of entries in the Socket connection table.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	1 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD063_Conf :**		
Name	SoAdTcpIpMainFunctionPeriod {SOAD_TCPIP_MainFunction_Period}		
Description	Determines the frequency at which the TcpIp_MainFunctionCyclic() is called in [s].		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	`0 .. INF`		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD006_Conf :**		
Name	SoAdTcpIpVersionInfoApi {TCPIP_VERSION_INFO_API}		
Description	Activates the TCPIP_GetVersionInfo API. TRUE: Enables the TCPIP_GetVersionInfo API. FALSE: TCPIP_GetVersionInfo API is not included.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD059_Conf :**		
Name	SoAdUdpNmApiEnabled {SOAD_UDPNM_API_ENABLED}		
Description	Activates the configurable interfaces to be used by UdpNm. TRUE: Enables support for the UdpNm API. FALSE: UdpNm API is not included.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	

Scope / Dependency	

SWS Item	**SOAD004_Conf :**		
Name	SoAdVersionInfoApi {SOAD_VERSION_INFO_API}		
Description	Activates the SoAd_GetVersionInfo() API. TRUE: Enables the SoAd_GetVersionInfo() API. FALSE: SoAd_GetVersionInfo() API is not included.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD060_Conf :**		
Name	SoAdXcpApiEnabled {SOAD_XCP_API_ENABLED}		
Description	Activates the configurable interfaces to be used by Xcp. TRUE: Enables support for the Xcp API. FALSE: Xcp API is not included.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.2.4 Socket Connection Table

[SOAD014] ⌈Remote Port set to `0xFFFF` allows for any source port in the received packets.⌋ ( )

[SOAD045] ⌈Remote Port set to `0x0000` allows for the source port to be set upon establishment of the connection.⌋ ( )

[SOAD046] ⌈A local IP address set to `00:00:00:00` required for the TCP/IP stack to aquire an address (by DHCP or link local configuration) and shall be updated after the address is configured.⌋ ( )

[SOAD040] ⌈For `TCP` connections the remote port used after the connection is established shall be updated in the Socket Connection Table.⌋ ( )

[SOAD015] ⌈Each table entry represents a possible connection and shall be used to allocate resources.⌋ ( )

[SOAD012] ⌈If multiple TCP connections are to be allowed into a single local TCP Port, multiple table entries are required. ⌋ ( )

[SOAD013] ⌈If a TCP connection is marked as the initiator of a connection further incoming connections can only be accepted if another table entry is present. ⌋ ( )

[SOAD011] ⌈The SoAd stack shall reject incoming connects that do not have a match in this table. This holds true independendly of the listed AUTOSAR connector. ⌋ ( )

10.2.5 SoAdSocketConnection

SWS Item	**SOAD009_Conf :**
Container Name	SoAdSocketConnection{SoAd_SocketConnection}
Description	Information required to receive and transmit data via the TCP/IP stack on a particular connection.
Configuration Parameters	

SWS Item	**SOAD025_Conf :**		
Name	SoAdAutosarConnector {SOAD_AUTOSAR_CONNECTOR}		
Description	Connection point within the AUTOSAR stack for this socket connection Availability of protocol plug-ins. Entries in the Socket and PDU Routing Tables.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	Cdd	--	
	DoIP	--	
	PduR	--	
	Xcp	--	
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD027_Conf :**		
Name	SoAdPduHeaderEnable {SOAD_PDU_HEADER_ENABLE}		
Description	Enables the transmission of the PDU header (ID, length) on this TCP/IP connection. TRUE: Send PDU header before data FALSE: Send data only		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	

Scope / Dependency	

SWS Item	**SOAD029_Conf :**		
Name	SoAdPduProvideBufferEnable {SOAD_PDU_PROVIDEBUFFER_ENABLE}		
Description	Enables the use of TP style API towards the PDU Router for this PDU. Will trigger the calls to ProvideRxBuffer and ProvideTxBuffer respectively. TRUE: The TP stype API is to be used towards the PDU Router. FALSE: The IF style API is to be used towards the PDU Router.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD067_Conf :**		
Name	SoAdResourceManagementEnable {SOAD_RESOURCE_MANAGEMENT_ENABLE}		
Description	Enables the resource management option for this socket. May not be activated for UDP sockets in receive and not for DoIP sockets. TRUE: resource management option enabled FALSE: resource management option disabled		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD026_Conf :**		
Name	SoAdSocketAutosarApi {SOAD_SOCKET_AUTOSAR_API}		
Description	Enables the use of the AUTOSAR call-back API for this connection. TRUE: Use AUTOSAR call-back API FALSE: Use BSD Socket API Availability of the AUTOSAR Call-back API in the TCP/IP stack.		
Multiplicity	1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	

Scope / Dependency	

SWS Item	**SOAD016_Conf :**		
Name	SoAdSocketId {SOAD_SOCKET_ID}		
Description	The Socket ID is used as a reference to a particular connection when transferring data to and from the PDU Router.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD017_Conf :**		
Name	SoAdSocketLocalIpAddress {SOAD_SOCKET_LOCAL_IP_ADDRESS}		
Description	Local IP address used for this connection. Network configuration. Local and Remote Address need to be in the same subnet.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD018_Conf :**		
Name	SoAdSocketLocalPort {SOAD_SOCKET_LOCAL_PORT}		
Description	Local UDP or TCP port used for this connection.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD021_Conf :**	
Name	SoAdSocketProtocol {SOAD_SOCKET_PROTOCOL}	
Description	Specifies the transport protocol (UDP or TCP).	
Multiplicity	1	
Type	EcucEnumerationParamDef	
Range	TCP	--
	UDP	--

ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD019_Conf :**		
Name	SoAdSocketRemoteIpAddress {SOAD_SOCKET_REMOTE_IP_ADDRESS}		
Description	IP address where NM packets are being sent to.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD020_Conf :**		
Name	SoAdSocketRemotePort {SOAD_SOCKET_REMOTE_PORT}		
Description	Remote UDP or TCP port used for this connection.		
Multiplicity	0..1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD022_Conf :**		
Name	SoAdSocketTcpInitiate {SOAD_SOCKET_TCP_INITIATE}		
Description	Specifies the initiator for this TCP connection. This parameter is only relevant for TCP connections. It will not be defined for UDP sockets. TRUE: This TCP connection is initiated by this module. FALSE: This TCP connection is to be initiated in the listen mode.		
Multiplicity	0..1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	

Scope / Dependency	

SWS Item	**SOAD023_Conf :**		
Name	SoAdSocketTcpNoDelay {SOAD_SOCKET_TCP_NODELAY}		
Description	Specifies not to use the congestion control mechanism for this connection. This parameter is only relevant for TCP connections. It will not be defined for UDP sockets. TRUE: This TCP connection will NOT use congestion control. FALSE: This TCP connection will use congestion control.		
Multiplicity	0..1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD024_Conf :**		
Name	SoAdSocketUdpListenOnly {SOAD_SOCKET_UDP_LISTEN_ONLY}		
Description	Used to disable the transmit functionality on this UDP port. This parameter is only relevant for UDP connections. TRUE: This UDP port cannot transmit data FALSE: This UDP port can send and receive data		
Multiplicity	0..1		
Type	EcucBooleanParamDef		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency			

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

10.2.6 SoAdSocketRoute

SWS Item	**SOAD008_Conf :**
Container Name	SoAdSocketRoute{SoAd_Socket_Route}
Description	Describes the path of a PDU from a socket in the TCP/IP stack to the PDU Router after reception in the TCP/IP Stack.
Configuration Parameters	

SWS Item	**SOAD037_Conf :**		
Name	SoAdDestinationSduLength {SOAD_DESTINATION_SDU_LENGTH}		
Description	Length in bytes of the data contained in the PDU.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD069_Conf :**		
Name	SoAdRxIndicationUL		
Description	This parameter defines the name of the <User_RxIndication> in case that SoAdUserRxIndicationUL is configured to Cdd. If SoAdUserRxIndicationUL equals PduR, Xcp or UdpNm, the name of the <User_RxIndication> is fixed and this parameter is skipped. If SoAdUserRxIndicationUL equals CDD the name of the <User_RxIndication> is selectable.		
Multiplicity	0..1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD036_Conf :**		
Name	SoAdSourceId {SOAD_SOURCE_ID}		
Description	ID contained in the packet received on the TCP/IP connection if the PDU header option is enabled.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967296		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	

Scope / Dependency	

SWS Item	**SOAD068_Conf :**		
Name	SoAdUserRxIndicationUL		
Description	This parameter defines the upper layer (UL) module to which the indication of the successfully received SoAd PDU has to be routed via <User_SoAdRxIndication>. This <User_SoAdRxIndication> has to be invoked when the RX indication is received by the EthIf module.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	Cdd	PDU Router	
	PduR	PDU Router	
	UdpNm	PDU Router	
	Xcp	XCP	
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD038_Conf :**		
Name	SoAdDestinationPduRef {SOAD_DESTINATION_PDU}		
Description	Reference to the global PDU structure		
Multiplicity	1		
Type	Reference to [Pdu]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD035_Conf :**		
Name	SoAdSourceSocketRef {SOAD_SOURCE_SOCKET_ID}		
Description	Connection on which the PDU was received. This references an entry in the Socket Connection Table.		
Multiplicity	1		
Type	Reference to [SoAdSocketConnection]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.2.7 SoAdPduRoute

SWS Item	**SOAD007_Conf :**
Container Name	SoAdPduRoute{SoAd_Pdu_Route}
Description	Describes the path of a PDU from the PDU Router to the socket in the TCP/IP stack for transmission.
Configuration Parameters	

SWS Item	**SOAD033_Conf :**		
Name	SoAdDestinationId {SOAD_DESTINATION_ID}		
Description	ID to be sent on the TCP/IP connection if the PDU header option is enabled.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 4294967296		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD031_Conf :**		
Name	SoAdSourcePduId {SOAD_SOURCE_PDU}		
Description	PDU ID of the PDU coming from the PDU Router.		
Multiplicity	1		
Type	EcucIntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD032_Conf :**		
Name	SoAdSourceSduLength {SOAD_SOURCE_SDU_LENGTH}		
Description	Length in bytes of the SDU to be sent over the TCP/IP stack.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	

Scope / Dependency	

SWS Item	**SOAD071_Conf :**		
Name	SoAdTxConfirmationUL		
Description	This optional parameter defines the name of the <User_TxConfirmation> in case that SoAdUserTxConfirmationUL is configured to Cdd. If SoAdUserTxConfirmationUL equals PduR, Xcp or UdpNm, the name of the <User_TxConfirmation> is fixed and this parameter is skipped. If SoAdUserTxConfirmationUL equals Cdd, the name of the <User_TxConfirmation> is selectable.		
Multiplicity	0..1		
Type	EcucFunctionNameDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD070_Conf :**		
Name	SoAdUserTxConfirmationUL		
Description	This parameter defines the upper layer (UL) module to which the confirmation of the successfully transmitted SoAdSourcePduId has to be routed via the <User_SoAdTxConfirmation>. This <User_SoAdTxConfirmation> has to be invoked when the confirmation of the configured SoAdSourcePduId will be received by a Tx confirmation event from the EthIf module.		
Multiplicity	1		
Type	EcucEnumerationParamDef		
Range	Cdd	PDU Router	
	PduR	PDU Router	
	UdpNm	PDU Router	
	Xcp	XCP	
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD034_Conf :**		
Name	SoAdDestinationSocketRef {SOAD_DESTINATION_SOCKET_ID}		
Description	Connection on which the PDU is to be sent on, references the appropriate entry in the Socket Connection Table.		
Multiplicity	1		
Type	Reference to [SoAdSocketConnection]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	

Scope / Dependency	

SWS Item	**SOAD030_Conf :**		
Name	SoAdSourcePduRef		
Description	Reference to the global PDU structure		
Multiplicity	1		
Type	Reference to [Pdu]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.2.8 SoAdDoIpConfig

SWS Item	**SOAD050_Conf :**
Container Name	SoAdDoIpConfig{SoAd_DoIP_Config} [Multi Config Container]
Description	This container contains all global configuration parameters of the DoIP plug-in.
Configuration Parameters	

SWS Item	**SOAD051_Conf :**		
Name	SoAdDoIpAliveCheckResponseTime {SoAd_DoIPAliveCheckResponseTime}		
Description	This parameter specifies the maximum time that a DoIP entity shall wait for an Alive Check Response after sending an Alive Check Request.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	`0 .. INF`		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD065_Conf :**		
Name	SoAdDoIpControlTimeout {SoAd_DoIPControlTimeout}		
Description	This parameter specifies the maximum time that the test equipment waits for a response to a previously sent control command.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	`0 .. INF`		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	

Scope / Dependency	

SWS Item	**SOAD052_Conf :**		
Name	SoAdDoIpGenericInactiveTime {SoAd_DoIPGenericInactiveTime}		
Description	This parameter specifies the maximum time of inactivity on a TCP_DATA socket before it is closed.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	`0 .. INF`		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD053_Conf :**		
Name	SoAdDoIpHostNameOpt {SoAd_DoIPHostNameOpt}		
Description	Defines the <manufacturer specific> part of the "host name option". Note: WD ISO 13400 implicitly shows 3 parts to the Host Name Option: 1) It is required to start with "DoIP_" 2) There may be a static OEM specific part 3) There may be a dynamic vehicle specific part, e.g. VIN SoAdDoIpHostNameOpt contains parts 1) and 2) only.		
Multiplicity	1		
Type	EcucStringParamDef		
Default value	--		
maxLength	--		
minLength	--		
regularExpression	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE, VARIANT-LINK-TIME
	Link time	--	
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	**SOAD054_Conf :**		
Name	SoAdDoIpInitialInactiveTime {SoAd_DoIPInitialInactiveTime}		
Description	This parameter specifies the maximum time of inactivity directly after a TCP_DATA socket was established. After the specified time without Routing Activation, the TCP_DATA socket is closed.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	`0 .. INF`		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	

Scope / Dependency	

SWS Item	**SOAD066_Conf :**		
Name	SoAdDoIpResponseTimeout {SoAd_DoIPResponseTimeout}		
Description	This parameter specifies the maximum time after which a DoIP information request must have been processed and the corresponding response must have been sent by the DoIP entity, otherwise the request or the response must be considered lost.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	`0 .. INF`		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD055_Conf :**		
Name	SoAdDoIpVidAnnounceInterval {SoAd_DoIPVIDAnnounceInterval}		
Description	This timing parameter specifies the time between the Vehicle Announcement Messages that are sent by DoIP entities after a valid IP address was configured.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	`0 .. INF`		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD056_Conf :**		
Name	SoAdDoIpVidAnnounceMaxWait {SoAd_DoIPVIDAnnounceMaxWait}		
Description	Describes the maximum time a DoIP entity shall wait before sending an Vehicle Identification Response.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	`0 .. INF`		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	

Scope / Dependency	

SWS Item	**SOAD057_Conf :**		
Name	SoAdDoIpVidAnnounceMinWait {SoAd_DoIPVIDAnnounceMinWait}		
Description	Describes the minimum time a DoIP entity shall wait before sending an Vehicle Identification Response.		
Multiplicity	1		
Type	EcucFloatParamDef		
Range	`0 .. INF`		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD058_Conf :**		
Name	SoAdDoIpVidAnnounceNum {SoAd_DoIPVIDAnnounceNum}		
Description	Specifies the number of Vehicle Announcement messages, which the DoIP entity sends after a valid IP address has been configured.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
SoAdDoIpEid	1	A unique 6-byte DoIp Entity Identification (EID)

10.2.9 SoAdDoIpEid

SWS Item	**SOAD098_Conf :**
Container Name	SoAdDoIpEid{SoAd_DoIP_Eid}
Description	A unique 6-byte DoIp Entity Identification (EID)
Configuration Parameters	

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
SoAdDoIpEidByte	6	One byte of the DoIp Entity Identification (EID).

10.2.10 SoAdDoIpEidByte

SWS Item	**SOAD099_Conf :**
Container Name	SoAdDoIpEidByte{SoAd_DoIP_Eid_Byte}
Description	One byte of the DoIp Entity Identification (EID).
Configuration Parameters	

SWS Item	**SOAD100_Conf :**		
Name	SoAdDoIpEidByteIndex {SoAD_DoIP_EID_BYTE_INDEX}		
Description	Index of the Eid byte array.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 5		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

SWS Item	**SOAD101_Conf :**		
Name	SoAdDoIpEidByteValue {SoAD_DoIP_EID_BYTE_VALUE}		
Description	Byte Value at the SoAdDoIpEidByteIndex position in the Eid byte array.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME
	Post-build time	X	VARIANT-POST-BUILD
Scope / Dependency			

No Included Containers

10.2.11 SoAdDoIpRoute

SWS Item	**SOAD040_Conf :**
Container Name	SoAdDoIpRoute{SoAd_DoIp_Route}
Description	A SoAd_DoIP_Route allocates a PDU ID to a combination of a DoIP source and a DoIP target address.
Configuration Parameters	

SWS Item	**SOAD041_Conf :**		
Name	SoAdDoIpSourceAddress {SoAD_DoIP_SOURCE_ADDRESS}		
Description	The logical DoIP address of the source entitiy.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD042_Conf :**		
Name	SoAdDoIpTargetAddress {SoAd_DoIP_TARGET_ADDRESS}		
Description	The logical DoIP address of the target entity.		
Multiplicity	1		
Type	EcucIntegerParamDef		
Range	0 .. 65535		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD043_Conf :**		
Name	SoAdDoIpSocketConnectionRef		
Description	Reference to the used socket connection.		
Multiplicity	1		
Type	Reference to [SoAdSocketConnection]		
ConfigurationClass	***Pre-compile time***	X	VARIANT-PRE-COMPILE
	Link time	X	VARIANT-LINK-TIME, VARIANT-POST-BUILD
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.2.12 SoAdDemEventConnectionParameterRefs

SWS Item	**SOAD084_Conf :**
Container Name	SoAdDemEventConnectionParameterRefs
Description	Container for the references to DemEventParameter elements which shall be invoked using the API Dem_ReportErrorStatus API in case the corresponding error occurs. The EventId is taken from the referenced DemEventParameter's DemEventId value. The standardized errors are provided in the container and can be extended by vendor specific error references.
Configuration Parameters	

SWS Item	**SOAD085_Conf :**		
Name	SOAD_E_AGAIN		
Description	Reference to the DemEventParameter which shall be issued when the error "Resource temporarily unavailable" has occured.		
Multiplicity	0..1		
Type	Reference to [DemEventParameter]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD089_Conf :**		
Name	SOAD_E_CONNABORTED		
Description	Reference to the DemEventParameter which shall be issued when the error "Software caused connection abort" has occured.		
Multiplicity	0..1		
Type	Reference to [DemEventParameter]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD093_Conf :**		
Name	SOAD_E_CONNREFUSED		
Description	Reference to the DemEventParameter which shall be issued when the error "Connection refused" has occured.		
Multiplicity	0..1		
Type	Reference to [DemEventParameter]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD090_Conf :**		
Name	SOAD_E_CONNRESET		
Description	Reference to the DemEventParameter which shall be issued when the error "Connection reset by peer" has occured.		
Multiplicity	0..1		
Type	Reference to [DemEventParameter]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	

Scope / Dependency	

SWS Item	**SOAD094_Conf :**		
Name	SOAD_E_HOSTDOWN		
Description	Reference to the DemEventParameter which shall be issued when the error "Host is down" has occured.		
Multiplicity	0..1		
Type	Reference to [DemEventParameter]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD095_Conf :**		
Name	SOAD_E_HOSTUNREACH		
Description	Reference to the DemEventParameter which shall be issued when the error "Host is down" has occured.		
Multiplicity	0..1		
Type	Reference to [DemEventParameter]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD086_Conf :**		
Name	SOAD_E_NETDOWN		
Description	Reference to the DemEventParameter which shall be issued when the error "Network is down" has occured.		
Multiplicity	0..1		
Type	Reference to [DemEventParameter]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD088_Conf :**		
Name	SOAD_E_NETRESET		
Description	Reference to the DemEventParameter which shall be issued when the error "Network dropped connection on reset" has occured.		
Multiplicity	0..1		
Type	Reference to [DemEventParameter]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	

Scope / Dependency	

SWS Item	**SOAD087_Conf :**		
Name	SOAD_E_NETUNREACH		
Description	Reference to the DemEventParameter which shall be issued when the error "Network is unreachable" has occured.		
Multiplicity	0..1		
Type	Reference to [DemEventParameter]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD091_Conf :**		
Name	SOAD_E_NOTCONN		
Description	Reference to the DemEventParameter which shall be issued when the error "Socket is not connected" has occured.		
Multiplicity	0..1		
Type	Reference to [DemEventParameter]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD096_Conf :**		
Name	SOAD_E_PIPE		
Description	Reference to the DemEventParameter which shall be issued when the error "Broken pipe" has occured.		
Multiplicity	0..1		
Type	Reference to [DemEventParameter]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD097_Conf :**		
Name	SOAD_E_SDULENGTH		
Description	Reference to the DemEventParameter which shall be issued when the error "SDU length mismatch" has occured.		
Multiplicity	0..1		
Type	Reference to [DemEventParameter]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	

Scope / Dependency	

SWS Item	**SOAD092_Conf :**		
Name	SOAD_E_TIMEDOUT		
Description	Reference to the DemEventParameter which shall be issued when the error "Operation timed out" has occured.		
Multiplicity	0..1		
Type	Reference to [DemEventParameter]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.2.13 SoAdDemEventParameterRefs

SWS Item	**SOAD080_Conf :**
Container Name	SoAdDemEventParameterRefs
Description	Container for the references to DemEventParameter elements which shall be invoked using the API Dem_ReportErrorStatus API in case the corresponding error occurs. The EventId is taken from the referenced DemEventParameter's DemEventId value. The standardized errors are provided in the container and can be extended by vendor specific error references.
Configuration Parameters	

SWS Item	**SOAD081_Conf :**		
Name	SOAD_E_INTR		
Description	Reference to the DemEventParameter which shall be issued when the error "Interrupted system call" has occured.		
Multiplicity	0..1		
Type	Reference to [DemEventParameter]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

SWS Item	**SOAD082_Conf :**		
Name	SOAD_E_IO		
Description	Reference to the DemEventParameter which shall be issued when the error "Input/output error" has occured.		
Multiplicity	0..1		
Type	Reference to [DemEventParameter]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	

Scope / Dependency	

SWS Item	**SOAD083_Conf :**		
Name	SOAD_E_UPPERBUFF		
Description	Reference to the DemEventParameter which shall be issued when the error "No buffer available in upper layer" has occured.		
Multiplicity	0..1		
Type	Reference to [DemEventParameter]		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.3 Published Information

[SOAD010] ⌈The standardized common published parameters as required by BSW00402 in the General Requirements on Basic Software Modules [2] shall be published within the header file of this module and need to be provided in the BSW Module Description. The according module abbreviation can be found in the List of Basic Software Modules [19].⌋ ( )

Additional module-specific published parameters are listed below if applicable.

11 Changes from Release 4.0 Revision 1

11.1 Deleted SWS Items

SWS Item	*Rationale*
SOAD051	Replaced by SOAD287

11.2 Changed SWS Items

SWS Item	*Rationale*
SOAD206	
SOAD052	
SOAD104	
SOAD093	

12 Not applicable requirements

[SOAD296] ⌈These requirements are not applicable to this specification.⌋ (BSW170, BSW00375, BSW00416, BSW168, BSW00423, BSW00424, BSW00425, BSW00426, BSW00427, BSW00429, BSW00432, BSW00434, BSW00336, BSW00417, BSW161, BSW162, BSW005, BSW00415, BSW164, BSW00325, BSW00326, BSW160, BSW00413, BSW00347, BSW00307, BSW00335, BSW00410, BSW00314, BSW00328, BSW00312, BSW006, BSW00355, BSW00306, BSW00309, BSW00330, BSW00331, BSW172, BSW010, BSW00333, BSW00321, BSW00341, BSW00334)