MICROSAR Classic Socket Adaptor

Technical Reference

Version 20.1.1

Authors	visalr, viswmc, vismda, vispcn, vissem, visjsb, viseje, visgyv
Status	Released

Document Information

History

Author	Date	Version	Remarks
visalr	2008-11-20	1.0	Creation of document
visalr	2009-10-05	2.0	Tool based configuration
viswmc	2012-01-16	2.1	Call-back for Ethernet State Manager and minor changes
viswmc	2012-07-24	2.2	DoIP extensions; dynamic UDP port usage
vismda	2012-10-04	2.3	DaVinci Configurator Pro support; AUTOSAR 4 support
vismda	2013-07-10	2.4	Customer specific extensions
vismda	2014-05-23	3.0	Updated to AUTOSAR 4.1
vismda	2015-03-26	4.0	Updated to AUTOSAR 4.2
vismda	2015-05-16	5.0	Optimized UDP retry behavior, Added BSD Socket API
vispcn	2015-11-16	6.0	Support of post-build loadable
vismda	2016-03-23	6.1	Support of TLS client Trigger Transmit API with SduLength In/Out Description for shutdown mechanism
vismda	2016-04-28	6.2	Release of BSD-Socket API
vismda	2016-05-31	6.3	Extension of BSD Socket API to support SOME/IP-SD under Linux
vismda	2016-11-11	7.0	MainFunction splitting Optimized TP transmission Trigger Transmit API for SoAd_IfTransmit Optimized buffer handling for PDU fan-out
vismda	2017-01-23	7.1	Event Queues and Timeout Lists
vismda	2017-02-22	7.2	Support Buffer Size up to 128kB
vismda	2017-05-08	8.0	Updated component history
vismda	2017-05-30	8.1	Updated service IDs
vismda	2017-06-19	8.2	PDU reception verification Transmission on specific socket connection Forward socket connection on reception
vismda	2017-08-01	8.3	Updated API description
vismda	2017-08-28	8.4	Reworked critical section chapter
vismda	2018-03-19	9.0	Reworked TP-API description
vissem	2018-07-31	10.0	Support INTEGRITY
vismda	2018-08-22	10.1	Support VLAN priorities for Linux
vismda	2019-03-12	11.0	Updated version history
vismda	2019-03-22	12.0	Added services to write/read/get events for DHCP options

vismda	2019-06-25	12.1	Updated version history
vismda	2019-07-24	13.0	Added DoIPInt to the supported modules
vismda	2019-08-27	13.1	Added service to retrieve and reset measurement data
visjsb	2020-02-13	14.0	Added description of additional critical sections
vismda	2020-06-02	14.1	Updated version history
vismda	2020-06-18	14.2	Updated version history
visjsb	2020-07-14	14.3	Updated version history
viseje	2020-09-21	14.4	Support for disabling the broadcast/multicast reception for LINUX and QNX
visjsb, viseje	2020-10-28	15.0	Document the restrictions of SoAd initialization Remove BSD dependency from SoAd
vismda	2020-11-26	15.1	Rework include structure for Socket API
viseje	2020-12-21	15.2	Support keeping online of socket connections after transmission on automatic socket connection setup
viseje	2021-01-26	15.3	Document meta data deviations, Document call restriction of socket connection open service, Support SOME/IP and SD measurement data
viseje	2021-03-12	15.4	Separate event queues and timeout lists for partitions, Update reentrancy documentation, Separate buffer pools for partitions
viseje	2021-04-29	16.0	Measurement data for partitions
viseje	2021-06-08	16.1	(Force) release remote address
vismda	2021-07-13	16.2	Configurable TCP buffer segments
viseje	2021-08-18	16.3	Support of security events
vismda	2021-09-02	16.4	Support of new AUTOSAR TLS
viseje	2021-09-28	17.0	Configurable UDP checksum calculation
viseje, visgyv	2021-11-03	17.1	Data separation for multi-partition feature, Support of optimistic transmission fan-out, Update of Unit State
viseje	2022-02-01	17.2	Support socket specific MSL timeout
viseje	2022-02-16	17.3	Updated configurable UDP checksum calculation
viseje	2022-03-08	17.4	Product name updated to MICROSAR Classic
vismda, viseje	2022-05-13	17.5	Added a “Caution” for optimized TP transmission, Reworked critical section documentation
viseje, vismda	2022-10-05	18.0	Reworked shutdown mechanism documentation, Documented address resolution retry queue
viseje	2022-11-28	18.1	Reworked nPdu extension features, Reworked trigger transmit documentation
visgyv	2023-02-08	18.2	Updated SoAd_SetRemoteAddr API, Scope of Delivery and Critical Sections
viseje	2023-04-17	19.0	Support of post-build selectable
viseje	2023-06-20	19.1	Reworked documentation of preemptive tasks

viseje	2023-07-31	19.2	Support of timeout for TCP client connection attempts
visgyv	2023-10-04	19.3	Make Bmc usage configurable, Document nested calls of critical sections
visgyv	2023-11-17	20.0	Added files of TxSocketManager unit to scope of delivery
viseje, visgyv	2024-04-02	20.1	Documented SoCon Mode Change queue, Updated UDP checksum, Added event TLS close notify received, Updated SoAd_ForceReleaseRemoteAddr

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	Specification of Socket Adaptor	R19-11
[2]	AUTOSAR	Specification of Default Error Tracer	R19-11
[3]	AUTOSAR	Specification of Diagnostic Event Manager	R4.2.2
[4]	AUTOSAR	List of Basic Software Modules	R19-11
[5]	AUTOSAR	Specification of Basic Software Multicore Library	R19-11
[6]	AUTOSAR	Specification of Socket Adaptor	R20-11
[7]	AUTOSAR	Specification of Socket Adaptor	R21-11
[8]	Vector	Technical Reference TcpIp	See delivery
[9]	Vector	Technical Reference MemMap	See delivery
[10]	Vector	User Manual Post-Build Loadable	See delivery
[11]	Vector	User Manual Identity Manager	See delivery

Scope of the Document

This technical reference describes the general use of the Socket Adaptor basis software. Please refer to your Release Notes to get a detailed description of the platform (host, compiler) your Vector Ethernet Bundle has been configured for.

Caution
We have configured the programs in accordance with your specifications in the questionnaire. Whereas the programs do support other configurations than the one specified in your questionnaire, Vector´s release of the programs delivered to your company is expressly restricted to the configuration you have specified in the questionnaire.

Contents

Illustrations

Tables

1 Introduction

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

Supported Configuration Variants:	pre-compile, post-build	
Vendor ID:	SOAD_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	SOAD_MODULE_ID	56 decimal (according to ref. [4])

The Socket Adaptor provides communication between PDU based communication and socket based communication via TcpIp. Following key features are offered by the Socket Adaptor:

- Support of TCP and UDP sockets over lower module TcpIp
- Supports multiple socket connections per local socket to support multiple communication partners on the same local socket
- Control API for socket connections or automated socket connection handling by Socket Adaptor
- Independent reception (Socket Route) and transmission path (Pdu Route) on a socket connection
- Support of Interface (IF) and Transport Protocol (TP) PDUs for upper layers
- Generic upper layer configuration

Figure 1-1 provides a functional overview over Socket Adaptor and some examples of possible configuration variants.

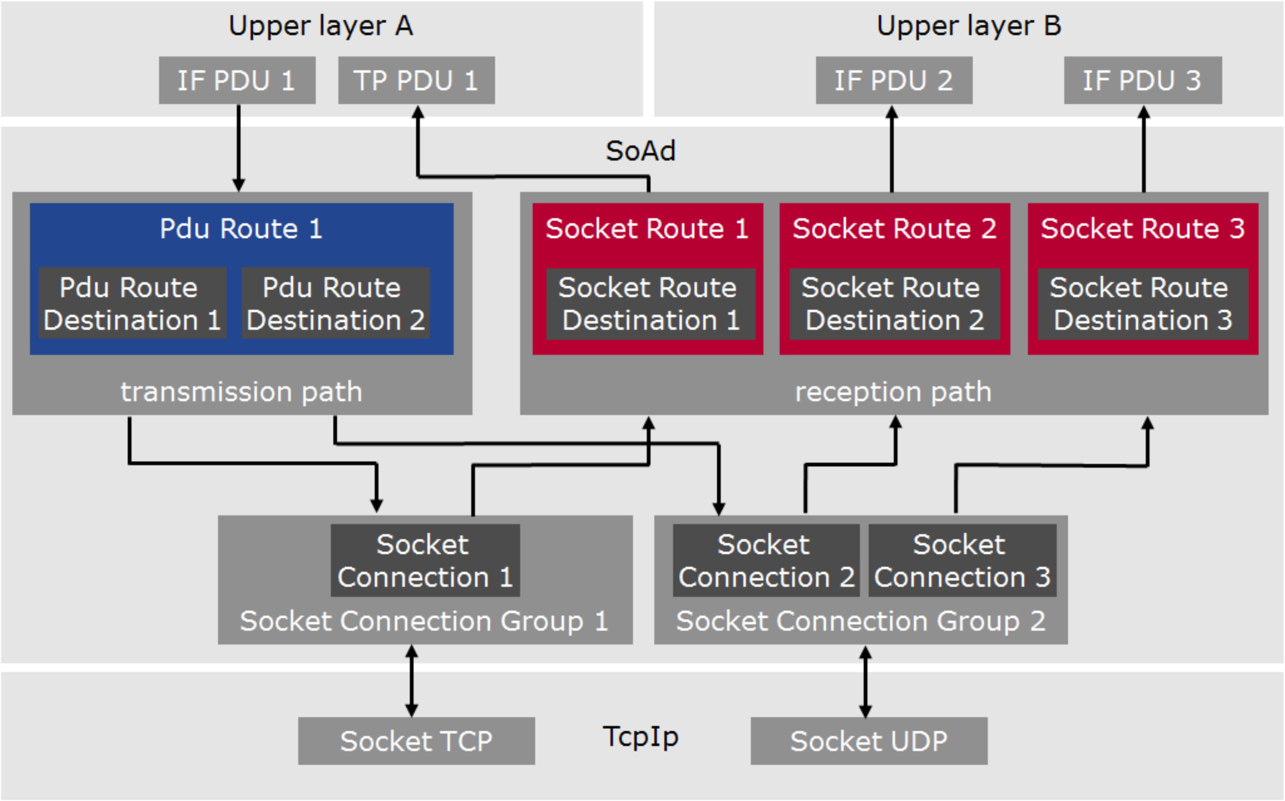
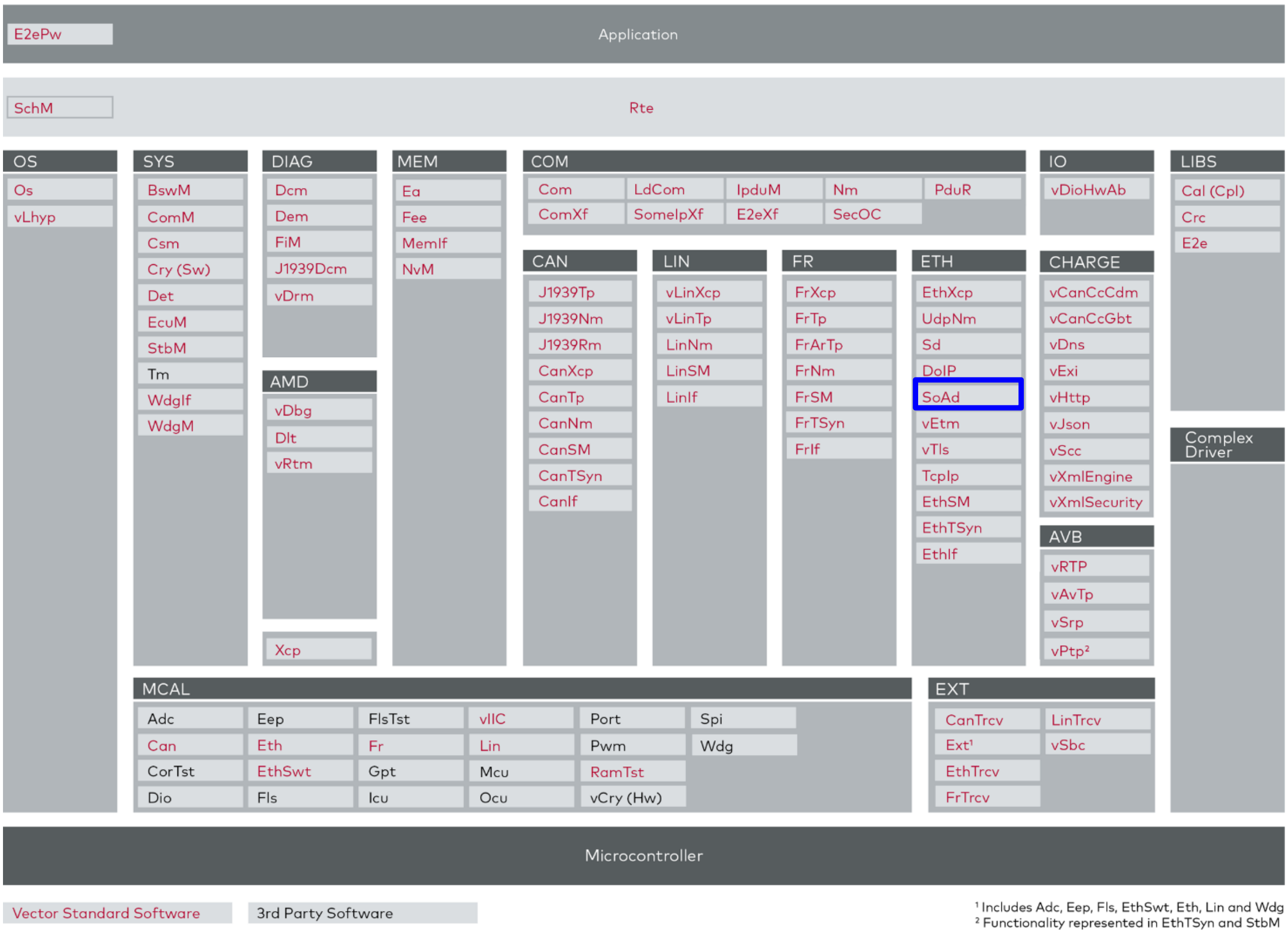


Figure 1-1 Functional Overview

1.1 Architecture Overview

The following figure shows where the Socket Adaptor is located in the AUTOSAR architecture.



[1] Includes Adc, Eep, Fls, EthSwt, Eth, Lin and Wdg
[2] Functionality represented in EthTSyn and StbM

Figure 1-2 AUTOSAR 4.2 Architecture Overview

The next figure shows the interfaces of the Socket Adaptor. These interfaces are described in chapter 4.

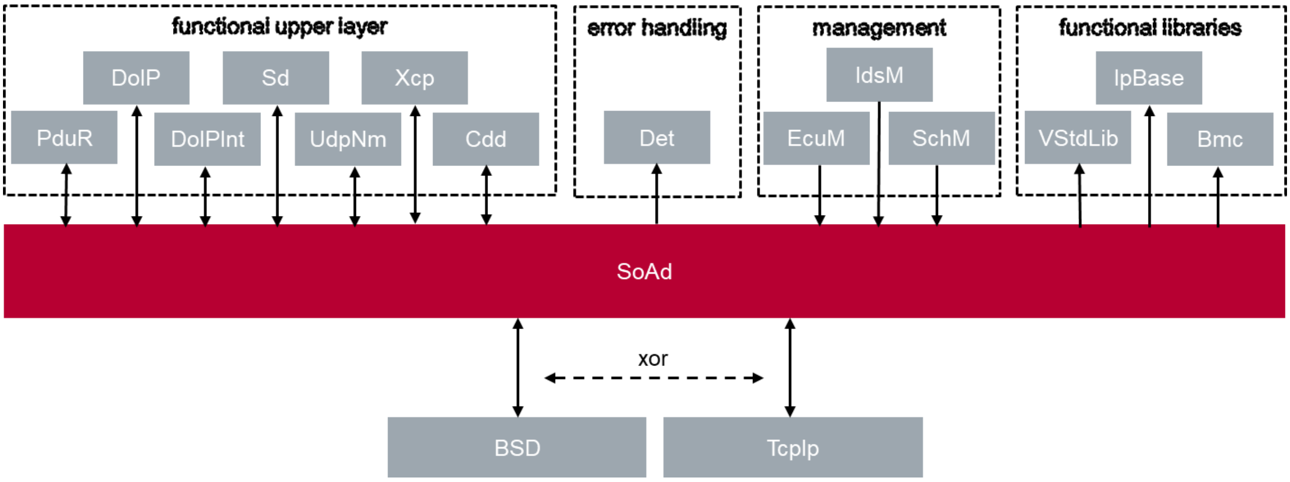


Figure 1-3 Interfaces to adjacent modules of the Socket Adaptor

Applications do not access the services of the BSW modules directly. They use the service ports provided by the BSW modules via the RTE. The Socket Adaptor does not support any service ports.

2 Functional Description

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

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

- Table 2-1 Supported AUTOSAR standard conform feature
- Table 2-2 Not supported AUTOSAR standard conform features

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

- Table 2-3 Features provided beyond the AUTOSAR standard

2.1 Features

The following features specified in [1] are supported:

Supported AUTOSAR Standard Conform Features
Socket Connections and Socket Connection Groups
PDU Transmission
PDU Reception
PDU Header option
Best Match Algorithm
Message Acceptance Policy
TP PDU Cancelation
Disconnection and recovery
Routing Groups
PDU fan-out
Buffer handling (e.g. nPdu feature)
Error handling
Version check
Address assignment services
Support of post-build loadable
Get and reset measurement data
Release remote IP address service
Support of post-build selectable

Table 2-1 Supported AUTOSAR standard conform feature

2.2 Deviations

The following features specified in [1] are not supported:

Category	Not Supported AUTOSAR Standard Conform Features
Functional	Socket Routes (TCP/UDP) with multiple TP upper layers and disabled PDU Header option
Functional	Socket Connection Open (Ch 7.1.1): Following ParamIds are not supported: - TCPIP_PARAMID_TCP_OPTIONFILTER - TCPIP_PARAMID_PATHMTU_ENABLE - TCPIP_PARAMID_FLOWLABEL - TCPIP_PARAMID_DSCP
Functional	Meta data is not provided via PduInfoPtr.MetaDataPtr. Instead, the meta data is provided after the payload data via PduInfoType.SduDataPtr.
Functional	SoAdSocketUdpAliveSupervisionTimeout: - Timeout gets reset on transmission, too - Timeout is not reset during SoAd_SetRemoteAddress
Functional	Development Errors ([1] Ch 7.12.1): SOAD_E_INV_METADATA
API	SoAd_TpCancelTransmit: ServiceIds from ASR-4.2.2 are used instead.
API	SoAd_TpCancelReceive: ServiceIds from ASR-4.2.2 are used instead.
API	SoAd_GetSoConMode
API	SoAd_TpChangeParameter
API	SoAd_RxIndication: Parameter “RemoteAddrPtr” and “BufPtr” are supported without const.
API	<Up>_[SoAd][If]TxConfirmation: parameter “result” is not supported
Config	SoAdSocketDifferentiatedServicesField
Config	SoAdSocketFlowLabel
Config	SoAdSocketPathMTUEnable
Config	SoAdSocketTcpTxQuota
Config	SoAdSocketTCPOptionFilterRef

Table 2-2 Not supported AUTOSAR standard conform features

2.2.1 Additions/ Extensions

The following features are provided beyond the AUTOSAR standard.

Features Provided Beyond The AUTOSAR Standard
Best Match Algorithm with PDU Header validation
Best Match Algorithm with Socket Route validation
UDP immediate IF transmission confirmation (TxConfirmation)
API extension to get the remote address of received data (SoAd_GetRcvRemoteAddr)
Additional change notifications
Configurable Socket API prefix
Shutdown mechanism

Features Provided Beyond The AUTOSAR Standard
MainFunction splitting
Optimized TP transmission
Trigger Transmit extensions
Queue size as trigger condition for nPdu feature
Mixed semantic support for nPdu feature
Event Queues and Timeout Lists
PDU reception verification
Read, write and events for DHCP options
Keeping online of socket connections with remote address set to wildcard after transmission on automatic socket connection setup
SOME/IP and SOME/IP SD service and method identifier measurement data
Multi-partition support
Force release of remote address
Configurable TCP buffer segments
Security Events according to [6]
Optimistic transmission fan-out
Socket specific MSL timeout
Timeout for TCP client connection attempts according to [7]

Table 2-3 Features provided beyond the AUTOSAR standard

! **Caution**
There may be also some other deviations which are not documented here.

2.2.2 Known Issues (low priority)

2.2.2.1 ESCAN00087305

Restricted functionality of compiler abstraction

The compiler abstraction for pointers does always use the identical 'ptrclass', independently from the memory location of the target. (It is not differentiated between variables stored in the pre-compile or post-build memory sections.)

Hence, the compiler abstraction cannot be used to specify and optimize pointers (would lead to compiler errors).

Workaround: Do not use special optimizations in compiler abstraction.

2.2.3 Hints

2.2.3.1 CDD Contribution Type

Socket Adaptor supports the “CddSoAdUpperLayerContribution” with schema according to AUTOSAR 4.0.3. Older versions support “CddComIfUpperLayerContribution” instead of “CddSoAdUpperLayerContribution”.

2.2.3.2 API deviation

The API to upper layer modules is implemented according to AUTOSAR 4.1.3 and partly to 4.2.1.

Please refer to chapter 4 for details.

2.2.3.3 UDP socket ressources bound at startup

If a UDP socket connection remote address contains wildcards, socket connection can be opened on reception according to [1]. To support this feature corresponding socket connection must bind a TcpIp socket at ECU startup (i.e. first MainFunction cycle).

2.2.3.4 SoAd_SetUniqueRemoteAddress() disables alive supervision timeout

If `SoAd_SetUniqueRemoteAddress()` is called for a UDP socket connection group and a corresponding socket connection in state online is found, alive supervision timeout will be disabled for this socket connection.

2.2.3.5 SoAd_CloseSoCon() if open/close counter is 0

If `SoAd_CloseSoCon()` is called with parameter `abort` set to `TRUE` and open/close counter is 0, caused by socket connection open in reception of data, the corresponding socket connection will be closed anyway. If parameter `abort` is set to `FALSE`, socket connection is not closed.

This behavior was implemented to close socket connections by user in all cases and to prevent always open socket connections that blocks communication with other remote entities.

2.3 Initialization

The Socket Adaptor is initialized via a `SoAd_InitMemory()` call followed by call of `SoAd_PreInit()`. Afterwards, `SoAd_Init()` must be called on each partition the Socket Adaptor is used on and initialization is finished by a call of `SoAd_PostInit()`. SoAd must not be initialized during runtime. The initialization is only allowed in uninit and shutdown state.

Example

```
SoAd_PreInit(SoAd_Config_Ptr);
SoAd_Init(SoAd_Config_Ptr);
SoAd_PostInit();
```

2.3.1 Configuration Variants 1, 2, 3 (Pre-Compile, Link-Time and Post-build selectable)

At configuration Variant 1 (Pre-compile), Variant 2 (Link-Time) and Variant 3 (Post-build selectable) the SoAd module has to be initialized using the `SoAd_PreInit()` and

SoAd_Init() function with the address of the pre-compile configuration data passed as parameter. The declaration of the pre-compile configuration data is contained in the files SoAd_Lcfg.h and SoAd_Lcfg.c. For Post-build selectable the pointer to the required variant data needs to be passed.

2.3.2 Configuration Variant 4 (Post-build loadable)

In this configuration Variant, the SoAd module has to be initialized using the SoAd_PreInit() and SoAd_Init() function with the address of the post-build configuration data passed as parameter. The declaration of the post-build configuration data is contained in the files SoAd_PBcfg.h and SoAd_PBcfg.c.

Please refer to chapter 5.1 to get information about supported configuration variants.

2.4 States

The Socket Adaptor has an extended state handling after calling the initialization functions (described in chapter before). Figure 2-1 shows the states of Socket Adaptor when using the shutdown feature described in 5.3.4.6.

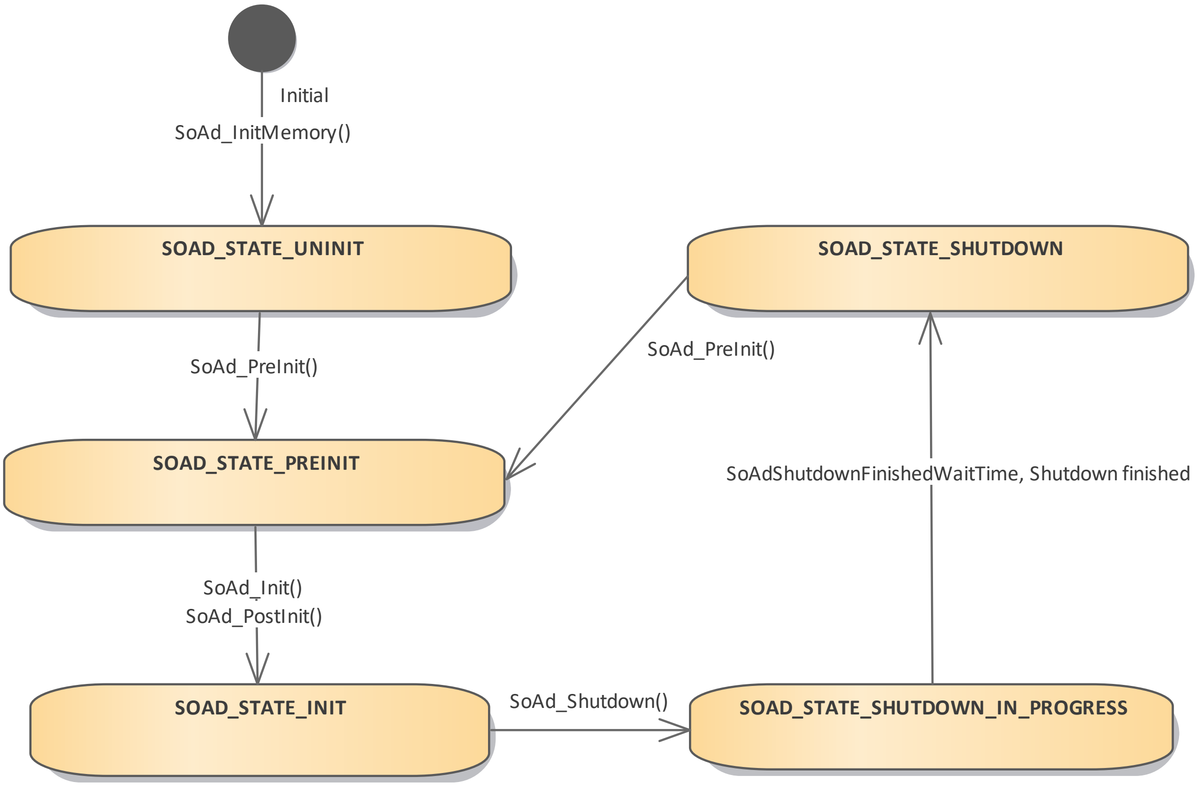


Figure 2-1 Module states

2.5 Main Functions

The Socket Adaptor has one main function (except when 5.3.4.7 MainFunction splitting is enabled) per SoAd instance (refer to 4.2.36 - 4.2.39 and to 5.3.4.18 for further information) which handles

- Socket connection state handling

- TP transmission/reception
- TP transmission/reception cancellation
- UDP nPduUdpTxBuffer
- TriggerTransmit transmission
- Handle pending transmission confirmation
- Release of remote addresses

Each main function only executes the functionality for the data paths and elements (e.g. socket connections) which is mapped to the calling instance.

2.6 Error Handling

2.6.1 Development Error Reporting

By default, development errors are reported to the DET using the service `Det_ReportError()` as specified in [2], if development error reporting is enabled (i.e. pre-compile parameter `SOAD_DEV_ERROR_DETECT==STD_ON`).

If another module is used for development error reporting, the function prototype for reporting the error can be configured by the integrator but must have the same signature as the service `Det_ReportError()`.

The reported Socket Adaptor ID is 56.

The reported service IDs identify the services which are described in 4.2. The following table presents the service IDs and the related services:

Service ID	Service
0x01	SOAD_SID_INIT
0x02	SOAD_SID_GET_VERSION_INFO
0x03	SOAD_SID_IF_TRANSMIT
0x04	SOAD_SID_TP_TRANSMIT
0x05	SOAD_SID_TP_CANCEL_TRANSMIT
0x06	SOAD_SID_TP_CANCEL_RECEIVE
0x07	SOAD_SID_GET_SO_CON_ID
0x08	SOAD_SID_OPEN_SO_CON
0x09	SOAD_SID_CLOSE_SO_CON
0x0A	SOAD_SID_REQ_IP_ADDR_ASSIGN
0x0B	SOAD_SID_RLS_IP_ADDR_ASSIGN
0x0C	SOAD_SID_GET_LOCAL_ADDR
0x0D	SOAD_SID_GET_PHYS_ADDR
0x0E	SOAD_SID_ENABLE_ROUTING

Service ID	Service
0x0F	SOAD_SID_DISABLE_ROUTING
0x10	SOAD_SID_SET_REMOTE_ADDR
0x11	SOAD_SID_TP_CHANGE_PARAMETER
0x12	SOAD_SID_RX_INDICATION
0x13	SOAD_SID_COPY_TX_DATA
0x14	SOAD_SID_TX_CONFIRMATION
0x15	SOAD_SID_TCP_ACCEPTED
0x16	SOAD_SID_TCP_CONNECTED
0x17	SOAD_SID_TCPIP_EVENT
0x18	SOAD_SID_LOCAL_IP_ADDR_ASSIGNMENT_CHG
0x19	SOAD_SID_MAIN_FUNCTION
0x1A	SOAD_SID_READ_DHCP_HOST_NAME_OPT
0x1B	SOAD_SID_WRITE_DHCP_HOST_NAME_OPT
0x1C	SOAD_SID_GET_REMOTE_ADDR
0x1D	SOAD_SID_IF_ROUT_GROUP_TRANSMIT
0x1E	SOAD_SID_SET_UNI_REMOTE_ADDR
0x1F	SOAD_SID_IF_SPEC_ROUT_GROUP_TRANSMIT
0x20	SOAD_SID_ENABLE_SPECIFIC_ROUTING
0x21	SOAD_SID_DISABLE_SPECIFIC_ROUTING
0x23	SOAD_SID_RELEASE_REMOTE_ADDR
0x45	SOAD_SID_GET_RESET_MEASURE_DATA
0xD0	SOAD_SID_MAIN_FUNCTION_RX
0xD1	SOAD_SID_MAIN_FUNCTION_STATE
0xD2	SOAD_SID_MAIN_FUNCTION_TX
0xD3	SOAD_SID_SHUTDOWN
0xD4	SOAD_SID_GET_RCV_REMOTE_ADDR
0xD5	SOAD_SID_GET_REMOTE_ADDR_STATE
0xD6	SOAD_SID_READ_DHCP_OPT
0xD7	SOAD_SID_WRITE_DHCP_OPT
0xD8	SOAD_SID_DHCP_EVENT
0xD9	SOAD_SID_FORCE_RELEASE_REMOTE_ADDR
0xDA	SOAD_SID_PRE_INIT
0xDB	SOAD_SID_POST_INIT

Service ID	Service
0xDC	SOAD_SID_SO_CON_MODE_CHG

Table 2-4 Service IDs

The development errors reported to DET are described in the following table:

Error Code	Description
0x00	SOAD_E_NO_ERROR
0x01	SOAD_E_NOTINIT
0x02	SOAD_E_PARAM_POINTER
0x03	SOAD_E_INV_ARG
0x04	SOAD_E_NOBUFS
0x05	SOAD_E_INV_PDUHEADER_ID
0x06	SOAD_E_INV_PDUID
0x07	SOAD_E_INV_SOCKETID
0x08	SOAD_E_INIT_FAILED
0x09	SOAD_E_INV_APPLICATION_ID
0x0A	SOAD_E_NO_PREINIT

Table 2-5 Development Errors reported to DET

2.6.2 Runtime Error Reporting

By default, runtime errors are reported to the DET using `Det_ReportRuntimeError()` as specified in [2], if the parameter `SoAdGeneral/SoAdRuntimeErrorReport` is enabled.

The runtime errors reported to DET are described in the following table:

Error Code	Description
0x10	SOAD_E_TCP_AUTOCONNECT_FAILED
0xD0	SOAD_E_NO_MODE_CHG_QUEUE_ELEM

Table 2-6 Runtime Errors reported to DET

2.6.3 Production Code Error Reporting

By default, production code related errors are reported to the DEM using the service `Dem_ReportErrorStatus()` as specified in [3].

The Socket Adaptor does not support DEM errors.

3 Integration

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

3.1 Scope of Delivery

The delivery of the Socket Adaptor consists out of these files:

File Name	Description
SoAd.c	Static source file
SoAd.h	Static header file
SoAd_Cbk.h	Static header file for callback functions
SoAd_EventQueue.c	Static source file for sub-module which handles event queues
SoAd_EventQueue.h	Static header file for sub-module which handles event queues
SoAd_Anomaly.c	Static source file for sub-module which handles reporting of anomalies (measurement data and security events)
SoAd_Anomaly.h	Static header file for sub-module which handles reporting of anomalies (measurement data and security events)
SoAd_BestMatch.c	Static source file for sub-module which handles best match algorithm
SoAd_BestMatch_Int.h	Static header file for internal API declaration of sub-module which handles best match algorithm
SoAd_Priv.h	Static header file for Socket Adaptor internal usage
SoAd_RouteGrp.c	Static source file for sub-module which handles routing groups
SoAd_RouteGrp.h	Static header file for sub-module which handles routing groups
SoAd_Rx.c	Static source file for sub-module which handles reception
SoAd_Rx.h	Static header file for sub-module which handles reception
SoAd_State.c	Static source file for sub-module which handles state
SoAd_State.h	Static header file for sub-module which handles state
SoAd_State_Int.h	Static header file for internal API declaration of sub-module which handles state
SoAd_SoCon.c	Static source file for sub-module which handles socket connections
SoAd_SoCon.h	Static header file for sub-module which handles socket connections
SoAd_LocalAddr.c	Static source file for sub-module which handles local address
SoAd_LocalAddr.h	Static header file for sub-module which handles local address
SoAd_LocalAddr_Int.h	Static header file for internal API declaration of sub-module which handles local address
SoAd_LocalAddr_Cbk.h	Static header file for callback functions of sub-module which handles local address
SoAd_RemoteAddr.c	Static source file for sub-module which handles remote address

File Name	Description
SoAd_RemoteAddr.h	Static header file for sub-module which handles remote address
SoAd_RemoteAddr_Int.h	Static header file for internal API declaration of sub-module which handles remote address
SoAd_TimeoutList.c	Static source file for sub-module which handles timeout lists
SoAd_TimeoutList.h	Static header file for sub-module which handles timeout lists
SoAd_Tx.c	Static source file for sub-module which handles transmission
SoAd_Tx.h	Static header file for sub-module which handles transmission
SoAd_TxSocketManager.c	Static source file for sub-module which handles transmission requests on sockets
SoAd_TxSocketManager_Int.h	Static header file for internal API declaration of sub-module which handles transmission requests on sockets
SoAd_TxSocketManager_Cbk.h	Static header file for callback functions of sub-module which handles transmission requests on sockets
SoAd_Util.c	Static source file for sub-module which provides general operations
SoAd_Util.h	Static header file for sub-module which provides general operations
SoAd_Types.h	Static header file containing types
SoAd_Lcfg.c	Generated source file (e.g. RAM/ROM mapping tables)
SoAd_Lcfg.h	Generated header file
SoAd_Lcfg_<OsAppl Shortname>.c	Partition-specific generated source file (e.g. RAM/ROM mapping tables of partition-specific data). Only for multi-partition solution.
SoAd_Lcfg_<OsAppl Shortname>.h	Partition-specific generated header file. Only for multi-partition solution.
SoAd_Cfg.h	Generated header file for configuration parameter (e.g. feature switches)
SoAd_PBcfg.c	Generated source file (Post-build configuration)
SoAd_PBcfg.h	Generated header file (Post-build configuration)
SoAd_PBcfg_<OsAppl Shortname>.c	Partition-specific generated source file (Post-build configuration). Only for multi-partition solution.
SoAd_PBcfg_<OsAppl Shortname>.h	Partition-specific generated header file (Post-build configuration). Only for multi-partition solution.
SoAd_GenTcpIpApi.h	Generated header file for the configured Socket API
SoAd_GenTypes.h	Generated header file containing generated types
SoAd_MemMap.h	Generated header file containing the memory sections of SoAd
SoAd.lib	Library if Socket Adaptor is not delivered with source code

Table 3-1 Implementation files

3.2 Critical Sections

All services and callbacks for transmission, reception and state changes of Socket Adaptor may be called in interrupt or task level. Thus, a synchronization mechanism is implemented to guarantee data consistency.

The synchronization mechanism defined by AUTOSAR covers the entering and leaving of so called critical sections.

The implementation of the critical sections must avoid that multiple relevant tasks or interrupt service routines can enter each of the critical sections more than once at the same time.

Relevant interrupt services in the Socket Adaptor context are interrupt services originated from physical bus events (Ethernet, CAN, LIN, FlexRay etc.).

Relevant tasks in the Socket Adaptor context are all tasks which call Socket Adaptor API functions. Usually, these tasks are limited to tasks on which other BSW modules (DoIP, Sd, Xcp etc.) are mapped to.

A critical section can be handled by using the so called “Exclusive Areas”. The Socket Adaptor defines the following exclusive areas:

- SOAD_EXCLUSIVE_AREA_0 is used whenever memory accesses must be protected from accesses of interrupting calls to services and callbacks of Socket Adaptor. This exclusive area may be entered in interrupt or task context. The frequency of entering and leaving this area will be very high. The average length of stay in the area is undefined but expected to be large since a lot of statements and complex code is handled. The critical section also covers external calls to `TcpIp_GetIpAddr()`.
- SOAD_EXCLUSIVE_AREA_1 is used in main function context to prevent that Socket Adaptor internal transmissions (only nPdu and routing group transmission) are interrupted by transmission requests of any user. The transmission requests are rejected in that case. This may lead to data loss if e.g. no retry mechanism is implemented by the SoAd users. The frequency of entering and leaving this area will be medium. The average length of stay in the area is undefined but expected to be large since the entire transmission path is protected. The critical section also covers external calls to `TcpIp_UdpTransmit()`, `TcpIp_TcpTransmit()` and `<Up>_[SoAd][If]TriggerTransmit()`.
- SOAD_EXCLUSIVE_AREA_MULTI_PARTITION is used whenever memory accesses must be protected from accesses of different partitions. This exclusive area may be entered in interrupt or task context. The frequency of entering and leaving this area will be low. The average length of stay in the area is low.

For an implementation of the critical sections SOAD_EXCLUSIVE_AREA_0 and SOAD_EXCLUSIVE_AREA_1 it could be sufficient to:

- Disable all bus relevant interrupts of all buses related to calls to Socket Adaptor API functions (e.g. gateway use-case).
- Disable all Ethernet bus relevant interrupts if all modules calling Socket Adaptor API functions are mapped to one task (e.g. SchM task) or a non-preemptive OS is used.

The implementation of the critical section SOAD_EXCLUSIVE_AREA_MULTI_PARTITION depends on whether multiple partitions are used.

- In case of a multi-partition use-case, it is required to implement this critical section as a spinlock.
  - It is required to map this critical section to another spinlock implementation than the TcpIp is using, if usage of Bmc is disabled. This is required since the TcpIp may call the SoAd while its spinlock is active.
- If the application runs on only one partition, the same mechanism as used for SOAD_EXCLUSIVE_AREA_0 shall be used.

Please note that these are only examples and that the actual implementation of the critical sections is highly dependent on the platform architecture and the system configuration.

Caution
Be aware that the Socket Adaptor calls all these critical sections nested (with themselves and other critical sections) but exits the critical sections in the reverse order they were entered. Note that SOAD_EXCLUSIVE_AREA_MULTI_PARTITION is only nested with the other critical sections. Please assert that this is supported by the implementation of the critical sections.

3.3 Main Functions

3.3.1 Preemption

This chapter describes what needs to be considered in case preemptive tasks are used since they may lead to interruptions of active call contexts.

SoAd expects that its own main functions do not interrupt each other. SoAd expects that an own main function does not interrupt the main functions of related modules and is not interrupted by main functions of related modules.

The SoAd related module is the TcpIp module.

Caution
Consider main function expectations when using preemptive tasks.

Side effects are expected if the transmission or reception context is interrupted by the main function context. In this case…

- …the socket connection state handling will be delayed to the next main function.
- …the API call sequence may be interrupted.
  - The transmission context (`SoAd_<If|Tp>Transmit`, `SoAd_If(Specific)RoutingGroupTransmit`, `SoAd_TpCancelTransmit`)

may be interrupted by `<Up>_[SoAd][If]TriggerTransmit`, `SoAd_CopyTxData` or `SoAd_TxConfirmation`.

- The reception context (`SoAd_TpCancelReceive`) may be interrupted by `<Up>_[SoAd][Tp]RxIndication`.

Caution
Be aware of the above-mentioned side effects when using preemptive tasks.

Additionally, please consider the usage of critical sections as described in chapter 3.2.

3.4 Memory Sections

The SoAd_MemMap.h file is generated by the MemMap Generator (`/ActiveEcuC/MemMap`). If adaptions should be done to the Memory Mapping of the SoAd, the changes must be configured in the MemMap Generator.

3.4.1 Memory Sections for Multi-partition use-case

If the Socket Adaptor is used in a multi-partition use-case, the RAM and ROM data will be generated into partition-specific memory sections.

The user must ensure that partitions cannot write into each other's memory (RO: Read only access). For partition-unspecific use-cases (e.g. measurement data), shared memory sections must be accessible (RW: read and write access) by all involved partitions. Table 3-2 gives an overview on the expected read and write accesses.

Memory Section	OsApplicationX	OsApplicationY
SOAD_OsApplicationX_<SectionType>	RW	RO
SOAD_OsApplicationY_<SectionType>	RO	RW
SOAD_<SectionType>	RW	RW

Table 3-2 Memory Mapping in multi-partition use-case

For the mapping of the partition-specific memory sections partition-specific software address methods (SwAddrMethods) are used. Refer to [9] for further details on SwAddrMethods. For the partition-specific SwAddrMethods the mapping is done automatically by the OS.

3.5 Multi-partition

When Socket Adaptor is configured in a multi-partition environment (refer to 5.3.4.18) freedom from interference (FFI) regarding memory cannot be guaranteed.

4 API Description

For an interfaces overview please see Figure 1-3.

4.1 Type Definitions

The types defined by the Socket Adaptor are described in this chapter.

Type Name	C-Type	Description	Value Range
SoAd_RemAddrStateType	uint8	Describes remote IP address and port state.	SOAD_SOCON_IP_SET_PORT_SET
			SOAD_SOCON_IP_SET_PORT_ANY
			SOAD_SOCON_IP_SET_PORT_NOT
			SOAD_SOCON_IP_ANY_PORT_SET
			SOAD_SOCON_IP_ANY_PORT_ANY
			SOAD_SOCON_IP_ANY_PORT_NOT
			SOAD_SOCON_IP_NOT_PORT_SET
			SOAD_SOCON_IP_NOT_PORT_ANY
			SOAD_SOCON_IP_NOT_PORT_NOT
SoAd_MeasurementIdxType	uint8	Index to select specific measurement data	SOAD_MEAS_DROP_TCP
			SOAD_MEAS_DROP_UDP
			SOAD_MEAS_DROP_TCP_CONNECTION
			SOAD_MEAS_DROP_UDP_SOCKET
			SOAD_MEAS_DROP_UDP_LENGTH
			SOAD_MEAS_INVALID_SOME_IP_SERVICE_ID
			SOAD_MEAS_INVALID_SOME_IP_METHOD_ID
			SOAD_MEAS_INVALID_SOME_IP_SD_SERVICE_ID
			SOAD_MEAS_INVALID_SOME_IP_SD_METHOD_ID
			SOAD_MEAS_ALL

Table 4-1 Type definitions

4.2 Services provided by Socket Adaptor

This chapter describes the service functions that are implemented by the Socket Adaptor and can be invoked by other modules. The prototypes of the service functions are provided in the header file `SoAd.h` by the Socket Adaptor.

4.2.1 SoAd_InitMemory

Prototype
void **SoAd_InitMemory** (void)

Parameter	
void	none
Return code	
void	none
Functional Description	
Initializes *_CLEARED_*-variables. Service to initialize module global variables at power up. This function initializes the variables in *_CLEARED_* sections. Used in case they are not initialized by the startup code.	
Particularities and Limitations	
Module is uninitialized.	
Call context	
> TASK > This function is Synchronous > This function is Non-Reentrant	

Table 4-2 SoAd_InitMemory

4.2.2 SoAd_PreInit

Prototype	
`void SoAd_PreInit (const SoAd_ConfigType *SoAdConfigPtr)`	
Parameter	
SoAdConfigPtr [in]	Configuration structure for initializing the module.
Return code	
void	none
Functional Description	
Pre-Initializes module. This function initializes the shared memory of SoAd module and sets the module to the pre-initialized state.	
Particularities and Limitations	
> Interrupts are disabled. SoAd_InitMemory has been called unless the *_CLEARED_*-variables are initialized by start-up code.	
Call context	
> TASK > This function is Synchronous > This function is Non-Reentrant > SoAd must not be initialized during runtime. The initialization is only allowed in uninit and shutdown state.	

Table 4-3 SoAd_PreInit

4.2.3 SoAd_Init

Prototype	
void **SoAd_Init** (const SoAd_ConfigType *SoAdConfigPtr)	
Parameter	
SoAdConfigPtr [in]	Configuration structure for initializing the module. This parameter is not used since it is already provided in context of SoAd_PreInit().
Return code	
void	none
Functional Description	
Initializes module. This function initializes the partition-specific memory of SoAd module of the application context this function is called in and sets the partition-specific initialized state.	
Particularities and Limitations	
> Interrupts are disabled. SoAd_PreInit() has been called.	
Call context	
> TASK > This function is Synchronous > This function is Non-Reentrant > SoAd must not be initialized during runtime. The initialization is only allowed in uninit and shutdown state. In multi-partition use-case: This function has to be called once in each partition context.	

Table 4-4 SoAd_Init

4.2.4 SoAd_PostInit

Prototype	
void **SoAd_PostInit** (void)	
Parameter	
void	none
Return code	
void	none
Functional Description	
Post-Initializes module. This function checks the initialization state of all partitions and sets the global initialized state.	
Particularities and Limitations	
> SoAd_Init() has been called on each partition.	
Call context	
> TASK > This function is Synchronous > This function is Non-Reentrant	

> SoAd must not be initialized during runtime. The initialization is only allowed in uninit and shutdown state. This function must be called only once and it must be called on the main partition in case of multi-partition.

Table 4-5 SoAd_PostInit

4.2.5 SoAd_IfTransmit

Prototype	
Std_ReturnType **SoAd_IfTransmit** (PduIdType SoAdSrcPduId, const PduInfoType *SoAdSrcPduInfoPtr)	
Parameter	
SoAdSrcPduId [in]	Tx PDU identifier.
SoAdSrcPduInfoPtr [in]	Pointer to PDU.
Return code	
Std_ReturnType	E_OK Transmit request was accepted.
Std_ReturnType	E_NOT_OK Transmit request was not accepted.
Functional Description	
Transmits an IF-PDU. -	
Particularities and Limitations	
-	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Reentrant for different PDUs and Non-Reentrant for the same PDU.	

Table 4-6 SoAd_IfTransmit

4.2.6 SoAd_IfRoutingGroupTransmit

Prototype	
Std_ReturnType **SoAd_IfRoutingGroupTransmit** (SoAd_RoutingGroupIdType id)	
Parameter	
id [in]	Routing group identifier.
Return code	
Std_ReturnType	E_OK Transmit request was accepted.
Std_ReturnType	E_NOT_OK Transmit request was not accepted.
Functional Description	
Triggers transmission of all IF-PDUs related to a routing group. Triggers transmission via trigger transmit in main function context.	
Particularities and Limitations	
-	

Call context
> TASK|ISR2 > This function is Reentrant

Table 4-7 SoAd_IfRoutingGroupTransmit

4.2.7 SoAd_IfSpecificRoutingGroupTransmit

Prototype	
`Std_ReturnType` **`SoAd_IfSpecificRoutingGroupTransmit`** `(SoAd_RoutingGroupIdType id, SoAd_SoConIdType SoConId)`	
Parameter	
id [in]	Routing group identifier.
SoConId [in]	Socket connection identifier.
Return code	
Std_ReturnType	E_OK Transmit request was accepted.
Std_ReturnType	E_NOT_OK Transmit request was not accepted.
Functional Description	
Triggers transmission of all IF-PDUs related to a routing group and socket connection. Triggers transmission via trigger transmit in main function context.	
Particularities and Limitations	
-	
Call context	
> TASK|ISR2 > This function is Reentrant	

Table 4-8 SoAd_IfSpecificRoutingGroupTransmit

4.2.8 SoAd_TpTransmit

Prototype	
`Std_ReturnType` **`SoAd_TpTransmit`** `(PduIdType SoAdSrcPduId, const PduInfoType *SoAdSrcPduInfoPtr)`	
Parameter	
SoAdSrcPduId [in]	Tx PDU identifier.
SoAdSrcPduInfoPtr [in]	Pointer to PDU (length is evaluated only).
Return code	
Std_ReturnType	E_OK Transmit request was accepted.
Std_ReturnType	E_NOT_OK Transmit request was not accepted.
Functional Description	
Transmits a TP-PDU. -	

Particularities and Limitations
-
Call context
> TASK|ISR2 > This function is Reentrant for different PDUs and Non-Reentrant for the same PDU.

Table 4-9 SoAd_TpTransmit

4.2.9 SoAd_Shutdown

Prototype	
Std_ReturnType **SoAd_Shutdown** (void)	
Parameter	
void	none
Return code	
Std_ReturnType	E_OK Shutdown request was accepted.
	SOAD_E_INPROGRESS Shutdown is in progress.
	E_NOT_OK Shutdown request was not accepted.
Functional Description	
Shuts down SoAd module. Closes all open socket connections and disables transmission and reception.	
Particularities and Limitations	
-	
Call context	
> TASK|ISR2 > This function is Non-Reentrant	

Table 4-10 SoAd_Shutdown

4.2.10 SoAd_TpCancelTransmit

Prototype	
Std_ReturnType **SoAd_TpCancelTransmit** (PduIdType PduId)	
Parameter	
PduId [in]	Tx PDU identifier.
Return code	
Std_ReturnType	E_OK Transmit cancellation request was accepted.
Std_ReturnType	E_NOT_OK Transmit cancellation request was not accepted.
Functional Description	
Requests transmission cancellation of a specific TP-PDU. -	

Particularities and Limitations
Transmission of PDU is requested via SoAd_TpTransmit.
Call context
> TASK|ISR2 > This function is Reentrant

Table 4-11 SoAd_TpCancelTransmit

4.2.11 SoAd_TpCancelReceive

Prototype	
`Std_ReturnType` **`SoAd_TpCancelReceive`** `(PduIdType PduId)`	
Parameter	
PduId [in]	Rx PDU identifier.
Return code	
Std_ReturnType	E_OK Receive cancellation request was accepted.
Std_ReturnType	E_NOT_OK Receive cancellation request was not accepted.
Functional Description	
Requests reception cancellation of a specific TP-PDU. -	
Particularities and Limitations	
Reception of PDU is initiated via <Up>_[SoAd][Tp]StartOfReception.	
Call context	
> TASK|ISR2 > This function is Reentrant	

Table 4-12 SoAd_TpCancelReceive

4.2.12 SoAd_GetSoConId

Prototype	
`Std_ReturnType` **`SoAd_GetSoConId`** `(PduIdType TxPduId, SoAd_SoConIdType *SoConIdPtr)`	
Parameter	
TxPduId [in]	Tx PDU identifier.
SoConIdPtr [out]	Pointer to the socket connection identifier.
Return code	
Std_ReturnType	E_OK Socket connection identifier was found.
Std_ReturnType	E_NOT_OK Socket connection identifier was not found.
Functional Description	
Returns the socket connection identifier of a specific Tx PDU identifier. -	

Particularities and Limitations
-
Call context
> TASK|ISR2 > This function is Synchronous > This function is Reentrant

Table 4-13 SoAd_GetSoConId

4.2.13 SoAd_OpenSoCon

Prototype	
`Std_ReturnType **SoAd_OpenSoCon** (SoAd_SoConIdType SoConId)`	
Parameter	
SoConId [in]	Socket connection identifier.
Return code	
Std_ReturnType	E_OK Open request was accepted.
Std_ReturnType	E_NOT_OK Open request was not accepted.
Functional Description	
Opens a socket connection. Opens the socket connection in context of main function.	
Particularities and Limitations	
-	
Call context	
> TASK|ISR2 > This function is Reentrant > The sum of the users must not call SoAd_OpenSoCon() more than 4294967295 ((2^32) - 1) times for a specific socket connection without calling SoAd_CloseSoCon().	

Table 4-14 SoAd_OpenSoCon

4.2.14 SoAd_CloseSoCon

Prototype	
`Std_ReturnType **SoAd_CloseSoCon** (SoAd_SoConIdType SoConId, boolean Abort)`	
Parameter	
SoConId [in]	Socket connection identifier.
Abort [in]	Flag to close socket connection immediately. [range: TRUE close immediately, FALSE close when open close sequence is 0]
Return code	
Std_ReturnType	E_OK Close request was accepted.
Std_ReturnType	E_NOT_OK Close request was not accepted.

Functional Description
Closes a socket connection. Closes the socket connection in context of main function.
Particularities and Limitations
-
Call context
> TASK|ISR2 > This function is Reentrant

Table 4-15 SoAd_CloseSoCon

4.2.15 SoAd_RequestIpAddrAssignment

Prototype	
Std_ReturnType **SoAd_RequestIpAddrAssignment** (SoAd_SoConIdType SoConId, SoAd_IpAddrAssignmentType Type, SoAd_SockAddrType *LocalIpAddrPtr, uint8 Netmask, SoAd_SockAddrType *DefaultRouterPtr)	
Parameter	
SoConId [in]	Socket connection identifier.
Type [in]	IP address type. [range in case of SOAD_SOCKET_API != SOAD_SOCKET_API_AUTOSAR: SOAD_IPADDR_ASSIGNMENT_STATIC .. SOAD_IPADDR_ASSIGNMENT_IPV6_ROUTER]
LocalIpAddrPtr [in]	Pointer to IP address which shall be assigned. [Points to one of the following structs depending on value of struct element domain: - SoAd_SockAddrInetType for IPv4 - SoAd_SockAddrInet6Type for IPv6]
Netmask [in]	Netmask in CIDR.
DefaultRouterPtr [in]	Pointer to default router (gateway) address. [Type == SOAD_IPADDR_ASSIGNMENT_STATIC: →DefaultRouterPtr != NULL_PTR Points to one of the following structs depending on configured IP address version of parameter SoConIdx: - SoAd_SockAddrInetType for IPv4 - SoAd_SockAddrInet6Type for IPv6 Type != SOAD_IPADDR_ASSIGNMENT_STATIC: → DefaultRouterPtr == NULL_PTR]
Return code	
Std_ReturnType	E_OK Assignment request was accepted.
Std_ReturnType	E_NOT_OK Assignment request was not accepted.
Functional Description	
Requests IP address assignment on a local address identified by a socket connection. -	
Particularities and Limitations	
-	

Call context
> TASK|ISR2 > This function is Synchronous > This function is Reentrant for different local IP addresses and Non-Reentrant for the same local IP address.

Table 4-16 SoAd_RequestIpAddrAssignment

4.2.16 SoAd_ReleaseIpAddrAssignment

Prototype	
`Std_ReturnType` **`SoAd_ReleaseIpAddrAssignment`** `(SoAd_SoConIdType SoConId)`	
Parameter	
SoConId [in]	Socket connection identifier.
Return code	
Std_ReturnType	E_OK Release request was accepted.
Std_ReturnType	E_NOT_OK Release request was not accepted.
Functional Description	
Releases IP address assignment on a local address identified by a socket connection. -	
Particularities and Limitations	
-	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Reentrant for different local IP addresses and Non-Reentrant for the same local IP address.	

Table 4-17 SoAd_ReleaseIpAddrAssignment

4.2.17 SoAd_GetLocalAddr

Prototype	
`Std_ReturnType` **`SoAd_GetLocalAddr`** `(SoAd_SoConIdType SoConId, SoAd_SockAddrType *LocalAddrPtr, uint8 *NetmaskPtr, SoAd_SockAddrType *DefaultRouterPtr)`	
Parameter	
SoConId [in]	Socket connection identifier.
LocalAddrPtr [out]	Pointer to local address (IP and Port). [Points to one of the following structs depending on value of struct element domain: - SoAd_SockAddrInetType for IPv4 - SoAd_SockAddrInet6Type for IPv6]
NetmaskPtr [out]	Pointer to network mask (CIDR Notation).
DefaultRouterPtr [out]	Pointer to default router (gateway). [Points to one of the following structs depending on configured IP address version of parameter SoConId:

	- SoAd_SockAddrInetType for IPv4 - SoAd_SockAddrInet6Type for IPv6]
Return code	
Std_ReturnType	E_OK Request was accepted.
Std_ReturnType	E_NOT_OK Request was not accepted.
Functional Description	
Returns a local IP address identified by a socket connection. -	
Particularities and Limitations	
-	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Reentrant	

Table 4-18 SoAd_GetLocalAddr

4.2.18 SoAd_GetPhysAddr

Prototype	
`Std_ReturnType **SoAd_GetPhysAddr** (SoAd_SoConIdType SoConId, uint8 *PhysAddrPtr)`	
Parameter	
SoConId [in]	Socket connection identifier.
PhysAddrPtr [out]	Pointer to physical address. [Points to a uint8 array of length 6]
Return code	
Std_ReturnType	E_OK Request was accepted.
Std_ReturnType	E_NOT_OK Request was not accepted.
Functional Description	
Returns the physical address (MAC address) of a local interface identified by a socket connection. -	
Particularities and Limitations	
-	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Reentrant	

Table 4-19 SoAd_GetPhysAddr

4.2.19 SoAd_GetRemoteAddr

Prototype	
`Std_ReturnType` **`SoAd_GetRemoteAddr`** `(SoAd_SoConIdType SoConId, SoAd_SockAddrType *IpAddrPtr)`	
Parameter	
SoConId [in]	Socket connection identifier.
IpAddrPtr [out]	Pointer to remote address. [Points to one of the following structs depending on configured IP address version of parameter SoConId: - SoAd_SockAddrInetType for IPv4 - SoAd_SockAddrInet6Type for IPv6]
Return code	
Std_ReturnType	E_OK Request was accepted.
Std_ReturnType	E_NOT_OK Request was not accepted.
Functional Description	
Returns the remote address of a socket connection. -	
Particularities and Limitations	
-	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Reentrant	

Table 4-20 SoAd_GetRemoteAddr

4.2.20 SoAd_GetRemoteAddrState

Prototype	
`Std_ReturnType` **`SoAd_GetRemoteAddrState`** `(SoAd_SoConIdType SoConId, SoAd_SockAddrType *IpAddrPtr, SoAd_RemAddrStateType *RemAddrState)`	
Parameter	
SoConId [in]	Socket connection identifier.
IpAddrPtr [out]	Pointer to remote address. [Points to one of the following structs depending on configured IP address version of parameter SoConId: - SoAd_SockAddrInetType for IPv4 - SoAd_SockAddrInet6Type for IPv6]
RemAddrState [out]	Pointer to remote address state.
Return code	
Std_ReturnType	E_OK Request was accepted.
Std_ReturnType	E_NOT_OK Request was not accepted.

Functional Description
Returns the remote address and remote address state of a socket connection. -
Particularities and Limitations
-
Call context
> TASK|ISR2 > This function is Synchronous > This function is Reentrant

Table 4-21 SoAd_GetRemoteAddrState

4.2.21 SoAd_GetRcvRemoteAddr

Prototype	
Std_ReturnType **SoAd_GetRcvRemoteAddr** (SoAd_SoConIdType SoConId, SoAd_SockAddrType *IpAddrPtr)	
Parameter	
SoConId [in]	Socket connection identifier.
IpAddrPtr [out]	Pointer to remote address. [Points to one of the following structs depending on configured IP address version of parameter SoConId: - SoAd_SockAddrInetType for IPv4 - SoAd_SockAddrInet6Type for IPv6]
Return code	
Std_ReturnType	E_OK Request was accepted.
Std_ReturnType	E_NOT_OK Request was not accepted.
Functional Description	
Returns the remote address of the last received message on a socket connection. -	
Particularities and Limitations	
-	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Reentrant	

Table 4-22 SoAd_GetRcvRemoteAddr

4.2.22 SoAd_EnableRouting

Prototype
Std_ReturnType **SoAd_EnableRouting** (SoAd_RoutingGroupIdType id)

Parameter	
id [in]	Routing group identifier.
Return code	
Std_ReturnType	E_OK Request was accepted.
Std_ReturnType	E_NOT_OK Request was not accepted.
Functional Description	
Enables a routing group. -	
Particularities and Limitations	
-	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Reentrant	

Table 4-23 SoAd_EnableRouting

4.2.23 SoAd_EnableSpecificRouting

Prototype	
`Std_ReturnType` **`SoAd_EnableSpecificRouting`** `(SoAd_RoutingGroupIdType id, SoAd_SoConIdType SoConId)`	
Parameter	
id [in]	Routing group identifier.
SoConId [in]	Socket connection identifier.
Return code	
Std_ReturnType	E_OK Request was accepted.
Std_ReturnType	E_NOT_OK Request was not accepted.
Functional Description	
Enables a routing group on a specific socket connection. -	
Particularities and Limitations	
-	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Reentrant	

Table 4-24 SoAd_EnableSpecificRouting

4.2.24 SoAd_DisableRouting

Prototype	
Std_ReturnType **SoAd_DisableRouting** (SoAd_RoutingGroupIdType id)	
Parameter	
id [in]	Routing group identifier.
Return code	
Std_ReturnType	E_OK Request was accepted.
Std_ReturnType	E_NOT_OK Request was not accepted.
Functional Description	
Disables a routing group. -	
Particularities and Limitations	
-	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Reentrant	

Table 4-25 SoAd_DisableRouting

4.2.25 SoAd_DisableSpecificRouting

Prototype	
Std_ReturnType **SoAd_DisableSpecificRouting** (SoAd_RoutingGroupIdType id, SoAd_SoConIdType SoConId)	
Parameter	
id [in]	Routing group identifier.
SoConId [in]	Socket connection identifier.
Return code	
Std_ReturnType	E_OK Request was accepted.
Std_ReturnType	E_NOT_OK Request was not accepted.
Functional Description	
Disables a routing group on a specific socket connection. -	
Particularities and Limitations	
-	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Reentrant	

Table 4-26 SoAd_DisableSpecificRouting

4.2.26 SoAd_SetRemoteAddr

Prototype	
`Std_ReturnType` **`SoAd_SetRemoteAddr`** `(SoAd_SoConIdType SoConId, const SoAd_SockAddrType *RemoteAddrPtr)`	
Parameter	
SoConId [in]	Socket connection identifier.
RemoteAddrPtr [in]	Pointer to remote address. [Points to one of the following structs depending on configured IP address version of parameter SoConId: - SoAd_SockAddrInetType for IPv4 - SoAd_SockAddrInet6Type for IPv6]
Return code	
Std_ReturnType	E_OK Request was accepted.
Std_ReturnType	E_NOT_OK Request was not accepted.
Functional Description	
Sets the remote address of a socket connection. -	
Particularities and Limitations	
-	
Call context	
- TASK|ISR2 - This function is Synchronous - This function is Reentrant for different socket connection identifiers and Non-Reentrant for the same socket connection identifier. - The remote address cannot be set while transmission or reception is active.	

Table 4-27 SoAd_SetRemoteAddr

4.2.27 SoAd_SetUniqueRemoteAddr

Prototype	
`Std_ReturnType` **`SoAd_SetUniqueRemoteAddr`** `(SoAd_SoConIdType SoConId, const SoAd_SockAddrType *RemoteAddrPtr, SoAd_SoConIdType *AssignedSoConIdPtr)`	
Parameter	
SoConId [in]	Socket connection identifier (any socket connection in socket connection group).
RemoteAddrPtr [in]	Pointer to remote address. [Points to one of the following structs depending on configured IP address version of parameter SoConId: - SoAd_SockAddrInetType for IPv4 - SoAd_SockAddrInet6Type for IPv6]
AssignedSoConIdPtr [out]	Pointer to assigned socket connection identifier.

Return code	
Std_ReturnType	E_OK Request was accepted.
Std_ReturnType	E_NOT_OK Request was not accepted.
Functional Description	
Sets the remote address of a suitable socket connection in a socket connection group. Considers the best match algorithm to select the socket connection.	
Particularities and Limitations	
-	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Reentrant for different socket connection identifiers and Non-Reentrant for the same socket connection identifier.	

Table 4-28 SoAd_SetUniqueRemoteAddr

4.2.28 SoAd_ReleaseRemoteAddr

Prototype	
`Std_ReturnType` **`SoAd_ReleaseRemoteAddr`** `(SoAd_SoConIdType SoConId)`	
Parameter	
SoConId [in]	Socket connection identifier.
Return code	
void	none
Functional Description	
Releases the remote address (IP address and port) of the specified socket connection by setting it back to the configured remote address setting. Releases the remote address of a socket connection if it is not locked due to pending transmissions/receptions or an established TCP connection. In case of a locked connection the request is stored to process it later in main function context. Due to a high load of transmissions and reception of messages it may happen that a release of the remote address is not possible.	
Particularities and Limitations	
-	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Reentrant for different socket connection identifiers and Non-Reentrant for the same socket connection identifier.	

Table 4-29 SoAd_ReleaseRemoteAddr

4.2.29 SoAd_ForceReleaseRemoteAddr

Prototype	
`Std_ReturnType` **`SoAd_ForceReleaseRemoteAddr`** `(SoAd_SoConIdType SoConId)`	
Parameter	
SoConId [in]	Socket connection identifier.
Return code	
void	none
Functional Description	
Forces the release of the remote address (IP address and port) of the specified socket connection by setting it back to the configured remote address setting. Releases the remote address of a socket connection if it is not locked due to pending transmissions/receptions. If the connection is locked due to an open TCP connection, the remote address is released by forcing the socket connection to close. The closing behavior is configurable and leads to sending a TCP RST or TCP FIN. Additionally, new transmissions/receptions are rejected.	
Particularities and Limitations	
-	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Reentrant for different socket connection identifiers and Non-Reentrant for the same socket connection identifier.	

Table 4-30 SoAd_ForceReleaseRemoteAddr

4.2.30 SoAd_ReadDhcpHostNameOption

Prototype	
`Std_ReturnType` **`SoAd_ReadDhcpHostNameOption`** `(SoAd_SoConIdType SoConId, uint8 *length, uint8 *data)`	
Parameter	
SoConId [in]	Socket connection identifier.
length [in,out]	Length of buffer for hostname (length of provided buffer, updated to length of hostname).
data [out]	Pointer to buffer for hostname.
Return code	
Std_ReturnType	E_OK Request was accepted.
Std_ReturnType	E_NOT_OK Request was not accepted.
Functional Description	
Returns the DHCP hostname option currently configured on a local interface identified by a socket connection. -	

Particularities and Limitations
-
Call context
> TASK|ISR2 > This function is Synchronous > This function is Reentrant

Table 4-31 SoAd_ReadDhcpHostNameOption

4.2.31 SoAd_WriteDhcpHostNameOption

Prototype	
`Std_ReturnType` **`SoAd_WriteDhcpHostNameOption`** `(SoAd_SoConIdType SoConId, uint8 length, const uint8 *data)`	
Parameter	
SoConId [in]	Socket connection identifier.
length [in]	Length of buffer for hostname.
data [in]	Pointer to buffer for hostname.
Return code	
Std_ReturnType	E_OK Request was accepted.
Std_ReturnType	E_NOT_OK Request was not accepted.
Functional Description	
Sets the DHCP hostname option on a local interface identified by a socket connection. -	
Particularities and Limitations	
-	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Reentrant	

Table 4-32 SoAd_WriteDhcpHostNameOption

4.2.32 SoAd_ReadDhcpOption

Prototype	
`Std_ReturnType` **`SoAd_ReadDhcpOption`** `(SoAd_LocalAddrIdType IpAddrId, uint16 Option, uint16 *DataLengthPtr, uint8 *DataPtr)`	
Parameter	
IpAddrId [in]	IP address identifier.
Option [in]	DHCP option code.
DataLengthPtr [in,out]	Pointer to length of option buffer (updated to length of option on return).
DataPtr [out]	Pointer to option buffer.

Return code	
Std_ReturnType	E_OK Request was accepted.
Std_ReturnType	E_NOT_OK Request was not accepted.
Functional Description	
Returns a DHCP option on a local address. -	
Particularities and Limitations	
-	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Reentrant	

Table 4-33 SoAd_ReadDhcpOption

4.2.33 SoAd_WriteDhcpOption

Prototype	
`Std_ReturnType SoAd_WriteDhcpOption (SoAd_LocalAddrIdType IpAddrId, uint16 Option, uint16 DataLength, const uint8 *DataPtr)`	
Parameter	
IpAddrId [in]	IP address identifier.
Option [in]	DHCP option code.
DataLength [in]	Length of option buffer.
DataPtr [in]	Pointer to option buffer.
Return code	
Std_ReturnType	E_OK Request was accepted.
Std_ReturnType	E_NOT_OK Request was not accepted.
Functional Description	
Sets a DHCP option on a local address. -	
Particularities and Limitations	
-	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Reentrant	

Table 4-34 SoAd_WriteDhcpOption

4.2.34 SoAd_GetVersionInfo

Prototype	
`void SoAd_GetVersionInfo (Std_VersionInfoType *versioninfo)`	
Parameter	
versioninfo [out]	Pointer to where to store the version information. Parameter must not be NULL.
Return code	
void	none
Functional Description	
Returns the version information. Returns version information, vendor ID and AUTOSAR module ID of the component.	
Particularities and Limitations	
- Configuration Variant(s): SOAD_VERSION_INFO_API	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Reentrant	

Table 4-35 SoAd_GetVersionInfo

4.2.35 SoAd_GetAndResetMeasurementData

Prototype	
`void SoAd_GetAndResetMeasurementData (SoAd_MeasurementIdxType MeasurementIdx, boolean MeasurementResetNeeded, uint32 *MeasurementDataPtr)`	
Parameter	
MeasurementIdx [in]	The index to select specific measurement data. [range: SOAD_MEAS_DROP_TCP, SOAD_MEAS_DROP_UDP, SOAD_MEAS_DROP_TCP_CONNECTION, SOAD_MEAS_DROP_UDP_SOCKET, SOAD_MEAS_DROP_UDP_LENGTH, SOAD_MEAS_INVALID_SOME_IP_SERVICE_ID, SOAD_MEAS_INVALID_SOME_IP_METHOD_ID, SOAD_MEAS_INVALID_SOME_IP_SD_SERVICE_ID, SOAD_MEAS_INVALID_SOME_IP_SD_METHOD_ID, SOAD_MEAS_ALL]
MeasurementResetNeeded [in]	Flag to indicate if the counter needs to be reset. [range: TRUE, FALSE]
MeasurementDataPtr [out]	Buffer where the value of the counter is to be copied into. [range: POINTER may be NULL_PTR]

Return code	
Std_ReturnType	E_OK The operations were successful.
Std_ReturnType	E_NOT_OK The operations failed.
Functional Description	
Gets and Resets (if requested) the measurement data. Gets and Resets (if requested) the value of the counter of dropped TCP/UDP packets, dropped TCP connections, dropped UDP frames (wrong socket/length), invalid SOME/IP service identifiers, invalid SOME/IP method identifiers, invalid SOME/IP SD service identifiers and invalid SOME/IP SD method identifiers. The returned value contains the measurement data of all configured Socket Adaptor instances.	
Particularities and Limitations	
- Configuration Variant(s): SOAD_GET_RESET_MEASUREMENT_DATA_API	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Reentrant for different measurement indexes and Non-Reentrant for the same measurement index.	

Table 4-36 SoAd_GetAndResetMeasurementData

4.2.36 SoAd_MainFunctionRx

Prototype	
`void` **`SoAd_MainFunctionRx`** `(void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
Schedules the Socket Adaptor (Entry point for scheduling) and handles asynchronous reception.	
Particularities and Limitations	
In case that several SoAd instances exist, the `SoAd_MainFunctionRx()` exists per instance. As naming convention, the API name is in this case appended by the name of the instance: `SoAd_MainFunctionRx_<InstanceName>()`.	
Call context	
> TASK > This function is Synchronous > This function is Non-Reentrant	

Table 4-37 SoAd_MainFunctionRx

4.2.37 SoAd_MainFunctionState

Prototype	
`void SoAd_MainFunctionState (void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
Schedules the Socket Adaptor (Entry point for scheduling) and handles states.	
Particularities and Limitations	
In case that several SoAd instances exist, the `SoAd_MainFunctionState()` exists per instance. As naming convention, the API name is in this case appended by the name of the instance: `SoAd_MainFunctionState_<InstanceName>()`.	
Call context	
> TASK > This function is Synchronous > This function is Non-Reentrant	

Table 4-38 SoAd_MainFunctionState

4.2.38 SoAd_MainFunctionTx

Prototype	
`void SoAd_MainFunctionTx (void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
Schedules the Socket Adaptor (Entry point for scheduling) and handles asynchronous transmission.	
Particularities and Limitations	
In case that several SoAd instances exist, the `SoAd_MainFunctionTx()` exists per instance. As naming convention, the API name is in this case appended by the name of the instance: `SoAd_MainFunctionTx_<InstanceName>()`.	
Call context	
> TASK > This function is Synchronous > This function is Non-Reentrant	

Table 4-39 SoAd_MainFunctionTx

4.2.39 SoAd_MainFunction

Prototype	
`void **SoAd_MainFunction** (void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
Schedules the Socket Adaptor (Entry point for scheduling) and handles asynchronous reception and transmission and states.	
Particularities and Limitations	
In case that several SoAd instances exist, the `SoAd_MainFunction()` exists per instance. As naming convention, the API name is in this case appended by the name of the instance: `SoAd_MainFunction_<InstanceName>()`.	
Call context	
> TASK > This function is Synchronous > This function is Non-Reentrant	

Table 4-40 SoAd_MainFunction

4.3 Services used by Socket Adaptor

In the following table services provided by other components, which are used by the Socket Adaptor are listed. For details about prototype and functionality refer to the documentation of the providing component. Moreover, refer to chapter 5.3.4.5 for more information on the configuration of the TcpIp stack.

Component	API
DET	Det_ReportError
DET	Det_ReportRuntimeError
<TcpIp>	<TcpIp>_SoAdGetSocket
<TcpIp>	<TcpIp>_ChangeParameter
<TcpIp>	<TcpIp>_Bind
<TcpIp>	<TcpIp>_TcpListen
<TcpIp>	<TcpIp>_TcpConnect
<TcpIp>	<TcpIp>_TcpTransmit
<TcpIp>	<TcpIp>_UdpTransmit
<TcpIp>	<TcpIp>_TcpReceived
<TcpIp>	<TcpIp>_Close

Component	API
<TcpIp>	<TcpIp>_RequestIpAddrAssignment
<TcpIp>	<TcpIp>_ReleaseIpAddrAssignment
<TcpIp>	<TcpIp>_GetIpAddr
<TcpIp>	<TcpIp>_GetCtrlIdx
<TcpIp>	<TcpIp>_GetRemotePhysAddr
<TcpIp>	<TcpIp>_DhcpReadOption
<TcpIp>	<TcpIp>_DhcpWriteOption
<TcpIp>	<TcpIp>_DhcpV6ReadOption
<TcpIp>	<TcpIp>_DhcpV6WriteOption
IpBase	IPBASE_GET_UINT16
IpBase	IPBASE_GET_UINT32
IpBase	IPBASE_HTON16
VStdLib	VStdMemCpy
Bmc	Bmc_Load_u32
Bmc	Bmc_Store_u8
Bmc	Bmc_Store_u32
Bmc	Bmc_FetchAdd_u32
Bmc	Bmc_FetchSub_u32
IdsM	IdsM_SetSecurityEvent

Table 4-41 Services used by the Socket Adaptor

4.4 Callback Functions

This chapter describes the callback functions that are implemented by the Socket Adaptor and can be invoked by other modules. The prototypes of the callback functions are provided in the header file SoAd_Cbk.h by the Socket Adaptor.

4.4.1 SoAd_RxIndication

Prototype	
void **SoAd_RxIndication** (SoAd_SocketIdType SocketId, SoAd_SockAddrType *RemoteAddrPtr, uint8 *BufPtr, uint16 Length)	
Parameter	
SocketId [in]	Socket identifier.
RemoteAddrPtr [in]	Pointer to remote address. [Points to one of the following structs depending on value of struct element domain: - SoAd_SockAddrInetType for IPv4 - SoAd_SockAddrInet6Type for IPv6]

BufPtr [in]	Pointer to buffer of received data.
Length [in]	Length of received data.
Return code	
void	none
Functional Description	
Receives data from sockets. -	
Particularities and Limitations	
-	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Reentrant for different socket identifiers and Non-Reentrant for the same socket identifier.	

Table 4-42 SoAd_RxIndication

4.4.2 SoAd_CopyTxData

Prototype	
`BufReq_ReturnType` **`SoAd_CopyTxData`** `(SoAd_SocketIdType SocketId, uint8 *BufPtr, uint16 *BufLengthPtr)`	
Parameter	
SocketId [in]	Socket identifier.
BufPtr [in]	Pointer to buffer of provided transmission buffer.
BufLength|BufLengthPtr [in,out]	Pointer to length of provided transmission buffer.
Return code	
BufReq_ReturnType	BUFREQ_OK Copy request accepted.
BufReq_ReturnType	BUFREQ_E_NOT_OK Copy request not accepted.
Functional Description	
Copies data to provided transmission buffer. Uses "BufLengthPtr" to update length if less data is copied to provided buffer.	
Particularities and Limitations	
-	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Reentrant for different socket identifiers and Non-Reentrant for the same socket identifier.	

Table 4-43 SoAd_CopyTxData

4.4.3 SoAd_TxConfirmation

Prototype	
`void SoAd_TxConfirmation (SoAd_SocketIdType SocketId, uint16 Length)`	
Parameter	
SocketId [in]	Socket identifier.
Length [in]	Length of confirmed data.
Return code	
void	none
Functional Description	
Confirms transmission of data. -	
Particularities and Limitations	
-	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Reentrant for different socket identifiers and Non-Reentrant for the same socket identifier.	

Table 4-44 SoAd_TxConfirmation

4.4.4 SoAd_LocalIpAddrAssignmentChg

Prototype	
`void SoAd_LocalIpAddrAssignmentChg (SoAd_LocalAddrIdType IpAddrId, SoAd_IpAddrStateType State)`	
Parameter	
IpAddrId [in]	IP address identifier.
State [in]	State of IP address assignment.
Return code	
void	none
Functional Description	
Receives local IP address assignment state changes. -	
Particularities and Limitations	
-	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Reentrant for different IP address identifiers and Non-Reentrant for the same IP address identifier.	

Table 4-45 SoAd_LocalIpAddrAssignmentChg

4.4.5 SoAd_TcpAccepted

Prototype	
`Std_ReturnType **SoAd_TcpAccepted** (SoAd_SocketIdType SocketId, SoAd_SocketIdType SocketIdConnected, SoAd_SockAddrType *RemoteAddrPtr)`	
Parameter	
SocketId [in]	Listen socket identifier.
SocketIdConnected [in]	Connected socket identifier.
RemoteAddrPtr [in]	Pointer to remote addres. [Points to one of the following structs depending on value of struct element domain: - SoAd_SockAddrInetType for IPv4 - SoAd_SockAddrInet6Type for IPv6]
Return code	
Std_ReturnType	E_OK Connection was accepted.
Std_ReturnType	E_NOT_OK Connection was not accepted.
Functional Description	
Accepts TCP connections on a listen socket. -	
Particularities and Limitations	
-	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Reentrant for different socket identifiers and Non-Reentrant for the same socket identifier.	

Table 4-46 SoAd_TcpAccepted

4.4.6 SoAd_TcpConnected

Prototype	
`void **SoAd_TcpConnected** (SoAd_SocketIdType SocketId)`	
Parameter	
SocketId [in]	Socket identifier.
Return code	
void	none
Functional Description	
Handles TCP connections which have been initiated locally and are now successfully connected. -	

Particularities and Limitations
-
Call context
> TASK|ISR2 > This function is Synchronous > This function is Reentrant for different socket identifiers and Non-Reentrant for the same socket identifier.

Table 4-47 SoAd_TcpConnected

4.4.7 SoAd_TcpIpEvent

Prototype	
`void SoAd_TcpIpEvent (SoAd_SocketIdType SocketId, SoAd_EventType Event)`	
Parameter	
SocketId [in]	Socket identifier.
Event [in]	Event type. [SOAD_TCP_RESET, SOAD_TCP_CLOSED, SOAD_TCP_FIN_RECEIVED, SOAD_UDP_CLOSED, SOAD_TLS_HANDSHAKE_SUCCEEDED, SOAD_TLS_EVENT_CLOSENOTIFY_RECEIVED]
Return code	
void	none
Functional Description	
Handles events on sockets. -	
Particularities and Limitations	
-	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Reentrant for different socket identifiers and Non-Reentrant for the same socket identifier.	

Table 4-48 SoAd_TcpIpEvent

4.4.8 SoAd_DhcpEvent

Prototype	
`void SoAd_DhcpEvent (SoAd_LocalAddrIdType IpAddrId, SoAd_DhcpEventType Event)`	
Parameter	
IpAddrId [in]	IP address identifier.
Event [in]	Event type. [!= SOAD_DHCP_EVENT_INVALID]
Return code	
void	none

Functional Description
Notifies user of a DHCP event. -
Particularities and Limitations
-
Call context
> TASK|ISR2 > This function is Synchronous > This function is Reentrant

Table 4-49 SoAd_DhcpEvent

4.5 Configurable Interfaces

At its configurable interfaces the Socket Adaptor expects notification and callout functions which must be provided by the specific upper layer <Up> (e.g. PduR). The expected interface depends on configuration of each upper layer.

Availability, configuration dependencies and function prototypes are described in the following sub-chapters for each function.

4.5.1 <Up>_[SoAd][If]RxIndication

Prototype	
`void <Up>_[SoAd][If]RxIndication (PduIdType RxPduId, const PduInfoType* PduInfoPtr)`	
Parameter	
RxPduId [in]	Rx PDU identifier
PduInfoPtr [in]	Pointer to PDU
Return code	
void	none
Functional Description	
Receives IF-PDU.	
Particularities and Limitations	
- -	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Reentrant	

Table 4-50 <Up>_[SoAd][If]RxIndication

4.5.2 <Up>_[SoAd][If]TriggerTransmit

Prototype	
Std_ReturnType **<Up>_[SoAd][If]TriggerTransmit** (PduIdType TxPduId, PduInfoType* PduInfoPtr)	
Parameter	
TxPduId [in]	Tx PDU identifier
PduInfoPtr [in/out]	Pointer to PDU
Return code	
Std_ReturnType	E_OK Request was accepted.
Std_ReturnType	E_NOT_OK Request was not accepted.
Functional Description	
Copies data for a previously requested PDU via SoAd_IfTransmit if trigger transmit is used for the PDU.	
Particularities and Limitations	
- Available for upper layer with IF-API and enabled ‘SoAdRoutingGroupTxTriggerable’ for at least one routing group referenced in any ‘SoAdPduRoute’ or if ‘SoAdTxIfTriggerTransmit’ is enabled. PduInfoPtr->SduLength is set to provided buffer size and upper layer has to consider this value before copying to buffer.	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Reentrant	

Table 4-51 <Up>_[SoAd][If]TriggerTransmit

4.5.3 <Up>_[SoAd][If]TxConfirmation

Prototype	
void **<Up>_[SoAd][If]TxConfirmation** (PduIdType TxPduId)	
Parameter	
TxPduId [in]	Tx PDU identifier
Return code	
void	none
Functional Description	
Confirms transmission of an IF-PDU.	
Particularities and Limitations	
- Available for upper layer with IF-API and ‘SoAdIfTxConfirmation’ in ‘SoAdBswModules’ is enabled.	
Call context	
> TASK|ISR2 > This function is Synchronous	

> This function is Reentrant

Table 4-52 <Up>_[SoAd][If]TxConfirmation

4.5.4 <Up>_[SoAd][Tp]StartOfReception

Prototype	
`BufReq_ReturnType **<Up>_[SoAd][Tp]StartOfReception** (PduIdType RxPduId, PduInfoType* info, PduLengthType TpSduLength, PduLengthType* bufferSizePtr)`	
Parameter	
RxPduId [in]	Rx PDU identifier
info [in]	No used [range: NULL_PTR]
TpSduLength [in]	Total length of PDU to be received
bufferSizePtr [out]	Available receive buffer
Return code	
BufReq_ReturnType	BUFREQ_OK Reception request was accepted.
BufReq_ReturnType	BUFREQ_E_NOT_OK Reception request was not accepted.
Functional Description	
Starts reception of a TP-PDU.	
Particularities and Limitations	
- ‘BUFREQ_E_OVFL’ as return value is treated like ‘BUFREQ_E_NOT_OK’. Parameter ‘info’ can be configured to constant pointer by enabling ‘SoAdTpStartOfReceptionWithConstPointer’.	
Call context	
> TASK|ISR2 > This function is Reentrant	

Table 4-53 <Up>_[SoAd][Tp]StartOfReception

4.5.5 <Up>_[SoAd][Tp]CopyRxData

Prototype	
`BufReq_ReturnType **<Up>_[SoAd][Tp]CopyRxData** (PduIdType RxPduId, const PduInfoType* PduInfoPtr, PduLengthType* bufferSizePtr)`	
Parameter	
RxPduId [in]	Rx PDU identifier
PduInfoPtr [in]	Pointer to PDU
bufferSizePtr [out]	Available receive buffer
Return code	
BufReq_ReturnType	BUFREQ_OK Copy request was accepted.
BufReq_ReturnType	BUFREQ_E_NOT_OK Copy request was not accepted.

Functional Description
Copies received data of a TP-PDU.
Particularities and Limitations
- ‘BUFREQ_E_OVFL’ as return value is treated like ‘BUFREQ_E_NOT_OK’. Parameter ‘PduInfoPtr’ can be configured to constant pointer by enabling ‘SoAdTpCopyRxDataWithConstPointer’.
Call context
> TASK|ISR2 > This function is Reentrant

Table 4-54 <Up>_[SoAd][Tp]CopyRxData

4.5.6 <Up>_[SoAd][Tp]RxIndication

Prototype	
`void <Up>_[SoAd][Tp]RxIndication (PduIdType RxPduId, Std_ReturnType result)`	
Parameter	
RxPduId [in]	Rx PDU identifier
result [in]	Reception result
Return code	
void	none
Functional Description	
Indicates that a TP-PDU reception is finished.	
Particularities and Limitations	
- -	
Call context	
> TASK|ISR2 > This function is Reentrant	

Table 4-55 <Up>_[SoAd][Tp]RxIndication

4.5.7 <Up>_[SoAd][Tp]CopyTxData

Prototype	
`BufReq_ReturnType <Up>_[SoAd][Tp]CopyTxData (PduIdType TxPduId, const PduInfoType* PduInfoPtr, RetryInfoType* retry, PduLengthType* availableDataPtr)`	
Parameter	
TxPduId [in]	Tx PDU identifier
PduInfoPtr [in/out]	Pointer to PDU
retry [in]	Not used [range: NULL_PTR]

availableDataPtr [out]	Available transmission buffer
Return code	
BufReq_ReturnType	BUFREQ_OK Copy request was accepted.
BufReq_ReturnType	BUFREQ_E_NOT_OK Copy request was not accepted.
Functional Description	
Copies data to be transmitted of a TP-PDU.	
Particularities and Limitations	
- 'BUFREQ_E_OVFL' as return value is treated like 'BUFREQ_E_NOT_OK'. Parameter 'PduInfoPtr' can be configured to constant pointer by enabling 'SoAdTpCopyTxDataWithConstPointer'.	
Call context	
> TASK|ISR2 > This function is Reentrant	

Table 4-56 <Up>_[SoAd][Tp]CopyTxData

4.5.8 <Up>_[SoAd][Tp]TxConfirmation

Prototype	
`void <Up>_[SoAd][Tp]TxConfirmation (PduIdType TxPduId, Std_ReturnType result)`	
Parameter	
TxPduId [in]	Tx PDU identifier
result [in]	Transmission result
Return code	
void	none
Functional Description	
Confirms transmission of a TP-PDU.	
Particularities and Limitations	
- -	
Call context	
> TASK|ISR2 > This function is Reentrant	

Table 4-57 <Up>_[SoAd][Tp]TxConfirmation

4.5.9 <Up>_SoConModeChg

Prototype
`void <Up>_SoConModeChg (SoAd_SoConIdType SoConId, SoAd_SoConModeType Mode)`

Parameter	
SoConId [in]	Socket connection identifier
Mode [in]	New socket connection mode
Return code	
void	None
Functional Description	
Notifies about a socket connection mode change	
Particularities and Limitations	
- Available if ‘SoAdBswModules/SoAdSoConModeChg’ is enabled for an upper layer or if a ‘SoAdGeneral/SoAdAdditionalSoConModeChgCallback’ is configured.	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Reentrant	

Table 4-58 <Up>_SoConModeChg

4.5.10 <Up>_LocalIpAddrAssignmentChg

Prototype	
`void **<Up>_LocalIpAddrAssignmentChg** (SoAd_SoConIdType SoConId, SoAd_IpAddrStateType State)`	
Parameter	
SoConId [in]	Socket connection identifier
State [in]	New socket connection local IP address state
Return code	
void	None
Functional Description	
Notifies about IP assignment state for a specific socket connection.	
Particularities and Limitations	
- Available if ‘SoAdBswModules/SoAdLocalIpAddrAssigmentChg’ is enabled for an upper layer or if a ‘SoAdGeneral/SoAdAdditionalLocalIpAddrAssignmentChgCallback’ is configured.	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Reentrant	

Table 4-59 <Up>_LocalIpAddrAssignmentChg

4.5.11 <Up>_ShutdownFinished

Prototype	
`void <Up>_ShutdownFinished (void)`	
Parameter	
void	none
Return code	
void	None
Functional Description	
Notifies about finished shutdown.	
Particularities and Limitations	
- > Available if 'SoAdBswModules/SoAdShutdownFinishedCbk' is enabled for an upper layer.	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Reentrant	

Table 4-60 <Up>_ShutdownFinished

4.5.12 <Up_VerifyRxPdu>

Prototype	
`Std_ReturnType <Up_VerifyRxPdu> (const SoAd_SockAddrType * LocalAddrPtr, const SoAd_SockAddrType * RemoteAddrPtr, SoAd_PduHdrIdType PduHdrId, const PduInfoType * PduInfoPtr)`	
Parameter	
LocalAddrPtr [in]	Pointer to local socket address
RemoteAddrPtr [in]	Pointer to remote socket address
PduHdrId [in]	PDU Header ID
PduInfoPtr [in]	Pointer to PDU data [range: NULL_PTR if SoAdVerifyRxPduMaxDataLength == 0]
Return code	
Std_ReturnType	E_OK PDU reception verification succeeded
Std_ReturnType	E_NOT_OK PDU reception verification failed
Functional Description	
Verifies a PDU reception and indicates if a reception shall be continued or dropped.	
Particularities and Limitations	
- Configurable in 'SoAdVerifyRxPduCallback/SoAdVerifyRxPdu'	
Call context	
> TASK	

> This function is Synchronous > This function is Reentrant

Table 4-61 <Up_VerifyRxPdu>

4.5.13 <Up_DhcpEvent>

Prototype	
`void <Up_DhcpEvent> (SoAd_LocalAddrIdType IpAddrId, SoAd_DhcpEventType Event)`	
Parameter	
IpAddrId [in]	IP address identifier.
Event [in]	Event type. [!= TCPIP_DHCP_EVENT_INVALID]
Return code	
void	none
Functional Description	
Notifies user of a DHCP event. -	
Particularities and Limitations	
-	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Reentrant	

Table 4-62 <Up_DhcpEvent>

5 Configuration

There is one configuration tool to configure and generate the Socket Adaptor:

- DaVinci Configurator Classic

5.1 Configuration Variants

The Socket Adaptor supports the configuration variants

- `VARIANT-PRE-COMPILE`
- `VARIANT-POST-BUILD-LOADABLE`
- `VARIANT-POST-BUILD-SELECTABLE`

The configuration classes of the Socket Adaptor parameters depend on the supported configuration variants. For their definitions please see the SoAd_bswmd.arxml file.

5.2 Configuration of Post-Build

The configuration of post-build loadable is described in [10].

The configuration of post-build selectable is described in [11].

For a description which configuration parameters support post-build loadable or selectable please see the SoAd_bswmd.arxml file.

5.3 Configuration with DaVinci Configurator Classic

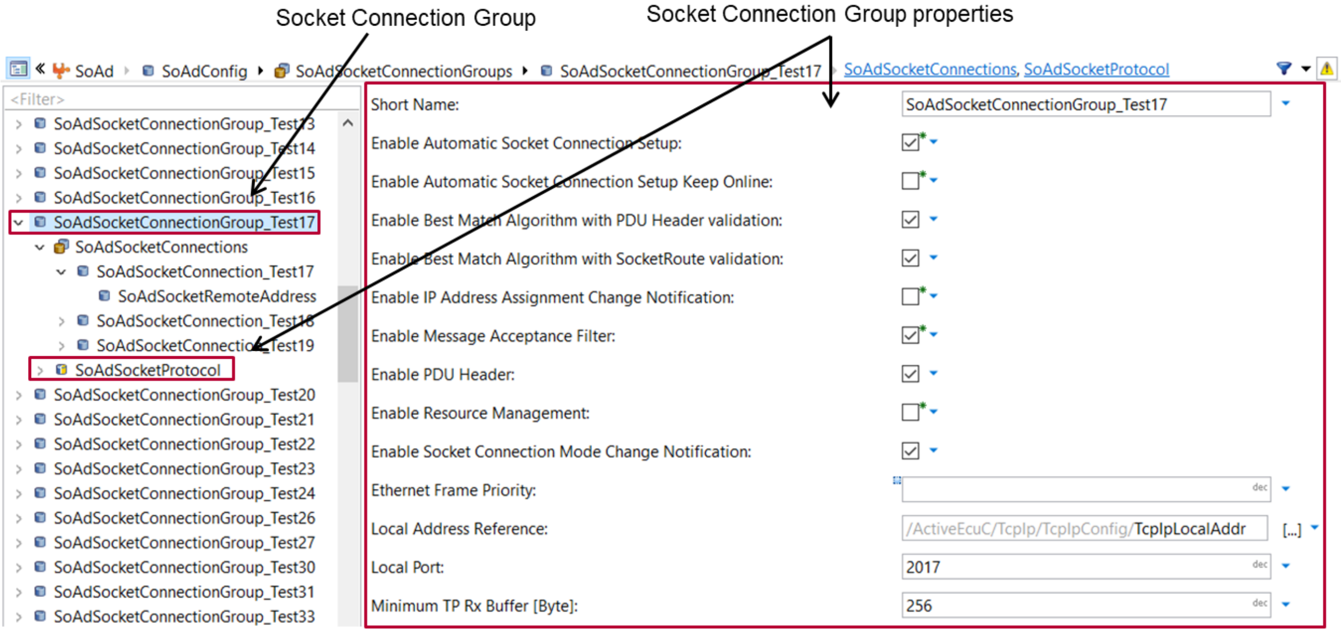
The Socket Adaptor is configured with the help of the configuration tool DaVinci Configurator Classic.

In the following sub-chapters some configuration hints are given to understand how to configure the Socket Adaptor correctly.

5.3.1 Socket Connection handling

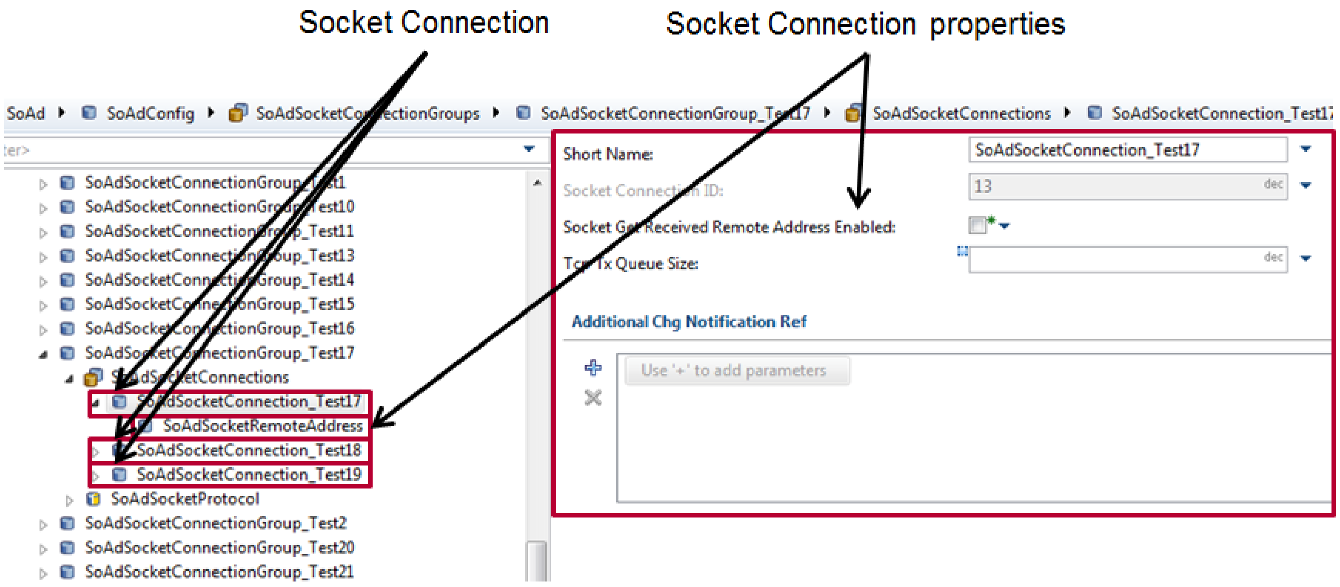
5.3.1.1 Socket Connection Group

To support multiple communication partners on one local socket (e.g. multiple clients on one server) the Socket Adaptor can define multiple socket connections. The general difference between these socket connections is the remote address (address of communication partner). All common properties are defined in socket connection group configuration (refer to Figure 5-1 and Figure 5-2).


Socket Connection Group
Socket Connection Group properties
SoAd ▸ SoAdConfig ▸ SoAdSocketConnectionGroups ▸ SoAdSocketConnectionGroup_Test17 ▸ SoAdSocketConnections, SoAdSocketProtocol
<Filter>
SoAdSocketConnectionGroup_Test13
SoAdSocketConnectionGroup_Test14
SoAdSocketConnectionGroup_Test15
SoAdSocketConnectionGroup_Test16
SoAdSocketConnectionGroup_Test17
SoAdSocketConnections
SoAdSocketConnection_Test17
SoAdSocketRemoteAddress
SoAdSocketConnection_Test18
SoAdSocketConnection_Test19
SoAdSocketProtocol
SoAdSocketConnectionGroup_Test20
SoAdSocketConnectionGroup_Test21
SoAdSocketConnectionGroup_Test22
SoAdSocketConnectionGroup_Test23
SoAdSocketConnectionGroup_Test24
SoAdSocketConnectionGroup_Test26
SoAdSocketConnectionGroup_Test27
SoAdSocketConnectionGroup_Test30
SoAdSocketConnectionGroup_Test31
SoAdSocketConnectionGroup_Test33

Short Name:	SoAdSocketConnectionGroup_Test17
Enable Automatic Socket Connection Setup:	☑
Enable Automatic Socket Connection Setup Keep Online:	□
Enable Best Match Algorithm with PDU Header validation:	☑
Enable Best Match Algorithm with SocketRoute validation:	☑
Enable IP Address Assignment Change Notification:	□
Enable Message Acceptance Filter:	☑
Enable PDU Header:	☑
Enable Resource Management:	□
Enable Socket Connection Mode Change Notification:	☑
Ethernet Frame Priority:	
Local Address Reference:	/ActiveEcuC/TcpIp/TcpIpConfig/TcpIpLocalAddr
Local Port:	2017
Minimum TP Rx Buffer [Byte]:	256

Figure 5-1 Socket Connection Group configuration


Socket Connection
Socket Connection properties
SoAd ▸ SoAdConfig ▸ SoAdSocketConnectionGroups ▸ SoAdSocketConnectionGroup_Test17 ▸ SoAdSocketConnections ▸ SoAdSocketConnection_Test17
ter>
SoAdSocketConnectionGroup_Test1
SoAdSocketConnectionGroup_Test10
SoAdSocketConnectionGroup_Test11
SoAdSocketConnectionGroup_Test13
SoAdSocketConnectionGroup_Test14
SoAdSocketConnectionGroup_Test15
SoAdSocketConnectionGroup_Test16
SoAdSocketConnectionGroup_Test17
SoAdSocketConnections
SoAdSocketConnection_Test17
SoAdSocketRemoteAddress
SoAdSocketConnection_Test18
SoAdSocketConnection_Test19
SoAdSocketProtocol
SoAdSocketConnectionGroup_Test2
SoAdSocketConnectionGroup_Test20
SoAdSocketConnectionGroup_Test21
Additional Chg Notification Ref
Use '+' to add parameters

Short Name:	SoAdSocketConnection_Test17
Socket Connection ID:	13
Socket Get Received Remote Address Enabled:	□
Tcp Tx Queue Size:	

Figure 5-2 Socket Connection configuration

In case of TCP and special cases of UDP buffers must be configured in the TcpIp module. The tool provides solving actions to add and optimize buffers.

5.3.1.2 Socket Connection establishment

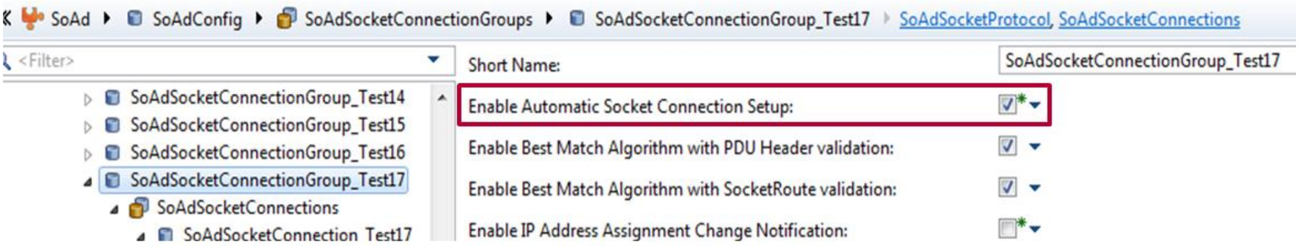
There are basically two different ways how to handle socket connections. Depending on the use-case following ways can be chosen:

- Manual
- Automatic

If an upper layer needs to control the socket connection state or remote address a manual socket connection establishment is recommended (e.g. DoIP). In all other cases Socket Adaptor can handle the socket connection on itself (e.g. PduR).

5.3.1.2.1 Manual

Manual socket connection establishment is enabled if the corresponding parameter is disabled (Figure 5-3).



SoAd ▸ SoAdConfig ▸ SoAdSocketConnectionGroups ▸ SoAdSocketConnectionGroup_Test17 ▸ SoAdSocketProtocol, SoAdSocketConnections

<Filter>

- SoAdSocketConnectionGroup_Test14
- SoAdSocketConnectionGroup_Test15
- SoAdSocketConnectionGroup_Test16
- SoAdSocketConnectionGroup_Test17
  - SoAdSocketConnections
    - SoAdSocketConnection_Test17

Short Name: SoAdSocketConnectionGroup_Test17

Enable Automatic Socket Connection Setup: ☑

Enable Best Match Algorithm with PDU Header validation: ☑

Enable Best Match Algorithm with SocketRoute validation: ☑

Enable IP Address Assignment Change Notification: □

Figure 5-3 Socket Connection Setup parameter

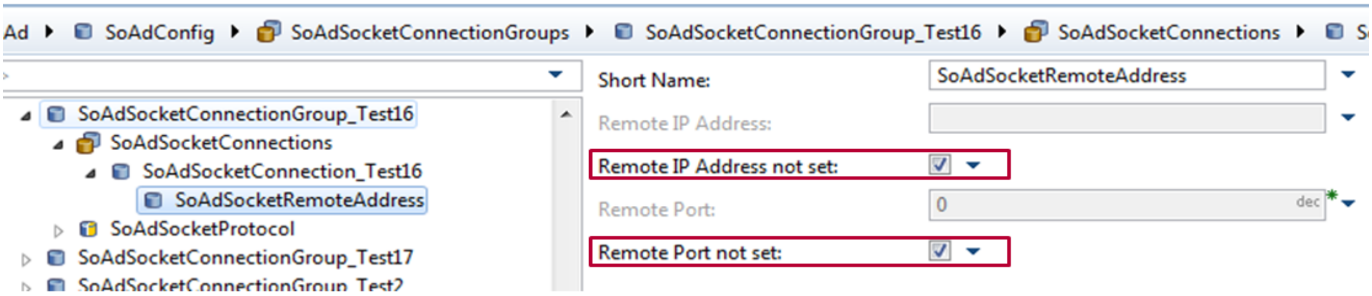
Following services are now available:

- SoAd_OpenSoCon()
- SoAd_CloseSoCon()
- SoAd_SetRemoteAddr()
- SoAd_SetUniqueRemoteAddr()

These services can be used to open and close a socket connection or set the remote address.

In case of UDP, the socket connection can be opened on reception if the message acceptance filter is enabled in the socket connection group and the remote address contains wildcards.

To set a port to wildcard, set port to 0. To configure an IP address to wildcard, empty the remote IP address field. Vector supports a special value for a not set IP address or port (Figure 5-4). If an IP address or port is not set, a socket connection is not considered in the best match algorithm (i.e. socket connection won't be opened on reception).



Ad ▸ SoAdConfig ▸ SoAdSocketConnectionGroups ▸ SoAdSocketConnectionGroup_Test16 ▸ SoAdSocketConnections ▸ S

- SoAdSocketConnectionGroup_Test16
  - SoAdSocketConnections
    - SoAdSocketConnection_Test16
      - SoAdSocketRemoteAddress
  - SoAdSocketProtocol
- SoAdSocketConnectionGroup_Test17
- SoAdSocketConnectionGroup_Test2

Short Name: SoAdSocketRemoteAddress

Remote IP Address:

Remote IP Address not set: ☑

Remote Port: 0 dec

Remote Port not set: ☑

Figure 5-4 Remote address "not set" parameter

5.3.1.2.2 Automatic

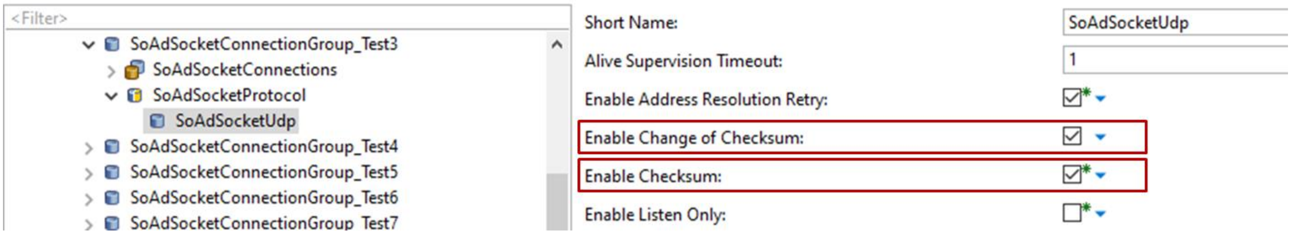
Automatic socket connection establishment is enabled if the corresponding parameter is enabled (Figure 5-3).

All services described in chapter 5.3.1.2.1 are not available and corresponding functions will return E_NOT_OK on call.

A socket connection is automatically opened if the remote address is set in configuration or on reception in case of UDP and the message acceptance filter is enabled in socket connection group.

5.3.1.3 Checksum handling on UDP sockets

It is possible to enable or disable the checksum calculation on UDP sockets. Figure 5-5 shows the configuration parameters. `SoAdSocketUdpChecksumChangeEnabled` specifies if the UDP checksum calculation setting shall be changed or not by calling `<TcpIp>_ChangeParameter()` API when opening the socket. `SoAdSocketUdpChecksumEnabled` represents the final setting of the UDP checksum calculation which is only applied when `SoAdSocketUdpChecksumChangeEnabled` is enabled. By default, the checksum calculation is enabled in TcpIp.



<Filter>
- SoAdSocketConnectionGroup_Test3
  - SoAdSocketConnections
  - SoAdSocketProtocol
    - SoAdSocketUdp
- SoAdSocketConnectionGroup_Test4
- SoAdSocketConnectionGroup_Test5
- SoAdSocketConnectionGroup_Test6
- SoAdSocketConnectionGroup_Test7

Short Name: SoAdSocketUdp
Alive Supervision Timeout: 1
Enable Address Resolution Retry: ☑
Enable Change of Checksum: ☑
Enable Checksum: ☑
Enable Listen Only: □

Figure 5-5 Configuration of checksum calculation on a UDP socket

Please refer to [8] for further information on how packets are handled on reception and transmission depending on the setting of this parameter.

Note

The checksum calculation is not configurable when

either Socket Adaptor is used together with vBsdAd module as lower layer,

or UDP frames are used via IPv6 (a checksum is mandatory in this case).

It is required to disable `SoAdSocketUdpChecksumChangeEnabled` in those cases.

If checksum offloading is done in hardware this parameter is usually expected to be disabled as the hardware could overwrite checksums of ‘0’ with a valid calculated checksum. This cannot be detected by software.

5.3.2 Transmission path

This chapter gives a short description about how configure a transmission path.

1. Configure a socket connection according to the use-case (chapter 5.3.1)
2. Add a PDU Route and choose referenced PDU and upper layer API type (Figure 5-6)

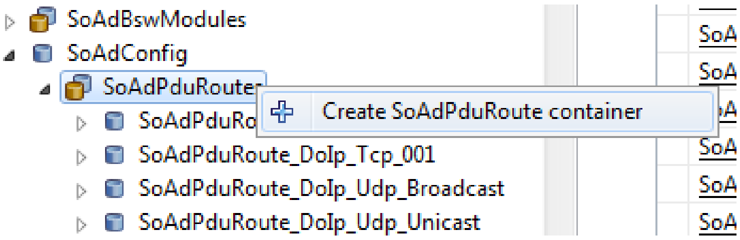

SoAdBswModules
SoAdConfig
SoAdPduRoutes
SoAdPduRo
Create SoAdPduRoute container
SoAdPduRoute_DoIp_Tcp_001
SoAdPduRoute_DoIp_Udp_Broadcast
SoAdPduRoute_DoIp_Udp_Unicast
SoA

Figure 5-6 Add PDU Route

3. Add one or multiple (fan-out) PDU Route Destination (Figure 5-7) and reference the socket connection created in 1

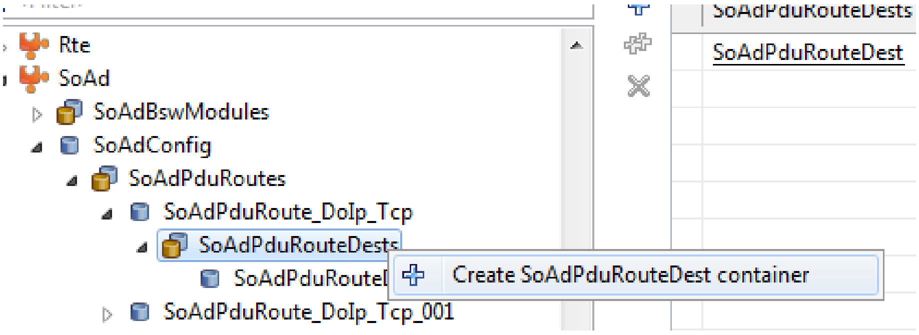


Figure 5-7 Add PDU Route Destination

5.3.3 Reception path

1. Configure a socket connection according to the use-case (chapter 5.3.1)
2. Add a Socket Route and reference socket connection created in 1(Figure 5-8)

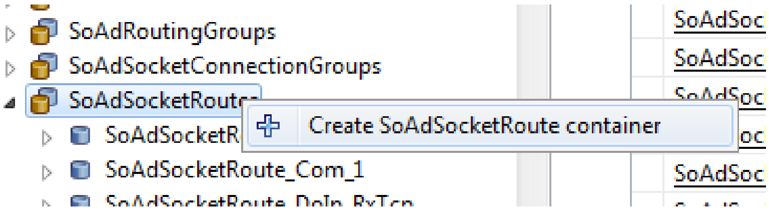


Figure 5-8 Add Socket Route

3. Add exactly one (fan-out not possible) Socket Route Destination and chose the upper layer PDU(Figure 5-9)

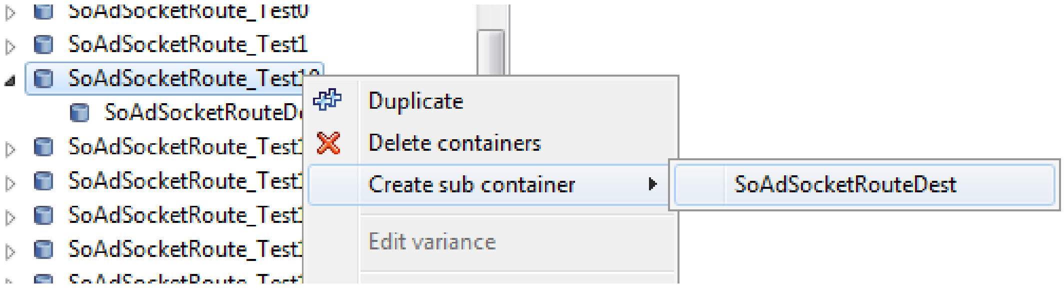


Figure 5-9 Add Socket Route Destination

5.3.4 Vector specific feature

5.3.4.1 Best Match Algorithm extensions

Best Match Algorithm according to [1] chooses a socket connection of a socket connection group considering remote addresses. In some cases it may be useful to consider more than remote address. Vector implements extensions of the Best Match Algorithm described here:

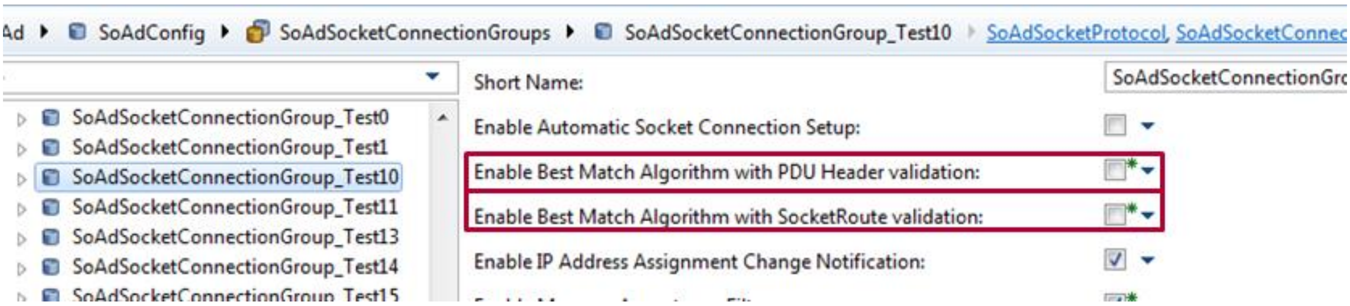
- Best Match Algorithm with PDU Header validation

  If this option is enabled PDU Header is extracted before socket connection is chosen. After PDU Header is received completely Best Match Algorithm is used to find the suitable socket connection. If the socket connection is found an additional check is performed to verify that the received PDU Header ID matches the configured value. In case of no match Best Match Algorithm will continue. If no socket connection and no socket route could be found the received PDU will be discarded.

- Best Match Algorithm with Socket Route validation

  This extension considers the existence of a socket route at a socket connection. If no socket route is configured for a socket connection the Best Match Algorithm will continue. This option can be used to prevent a socket connection setup on reception of data while other reception socket connections are used in same socket connection group.

Both described feature can be configured in the socket connection group (refer to Figure 5-10).



Ad ▸ SoAdConfig ▸ SoAdSocketConnectionGroups ▸ SoAdSocketConnectionGroup_Test10 ▸ SoAdSocketProtocol, SoAdSocketConnec

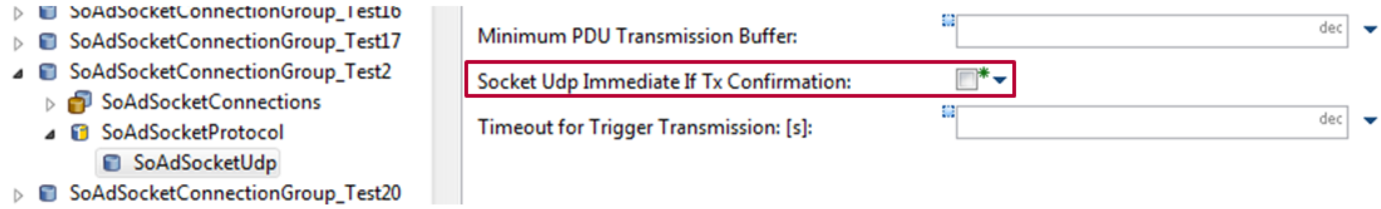
SoAdSocketConnectionGroup_Test0
SoAdSocketConnectionGroup_Test1
SoAdSocketConnectionGroup_Test10
SoAdSocketConnectionGroup_Test11
SoAdSocketConnectionGroup_Test13
SoAdSocketConnectionGroup_Test14
SoAdSocketConnectionGroup_Test15

Short Name: SoAdSocketConnectionGr
Enable Automatic Socket Connection Setup: □
Enable Best Match Algorithm with PDU Header validation: □
Enable Best Match Algorithm with SocketRoute validation: □
Enable IP Address Assignment Change Notification: ☑

Figure 5-10 Best Match Algorithm extensions

5.3.4.2 UDP Immediate IF TxConfirmaion

This feature is used to confirm data within interrupt level after successfully sending data. How to configure this feature is described in Figure 5-11.



SoAdSocketConnectionGroup_Test16
SoAdSocketConnectionGroup_Test17
SoAdSocketConnectionGroup_Test2
SoAdSocketConnections
SoAdSocketProtocol
SoAdSocketUdp
SoAdSocketConnectionGroup_Test20

Minimum PDU Transmission Buffer: dec
Socket Udp Immediate If Tx Confirmation: □
Timeout for Trigger Transmission: [s]: dec

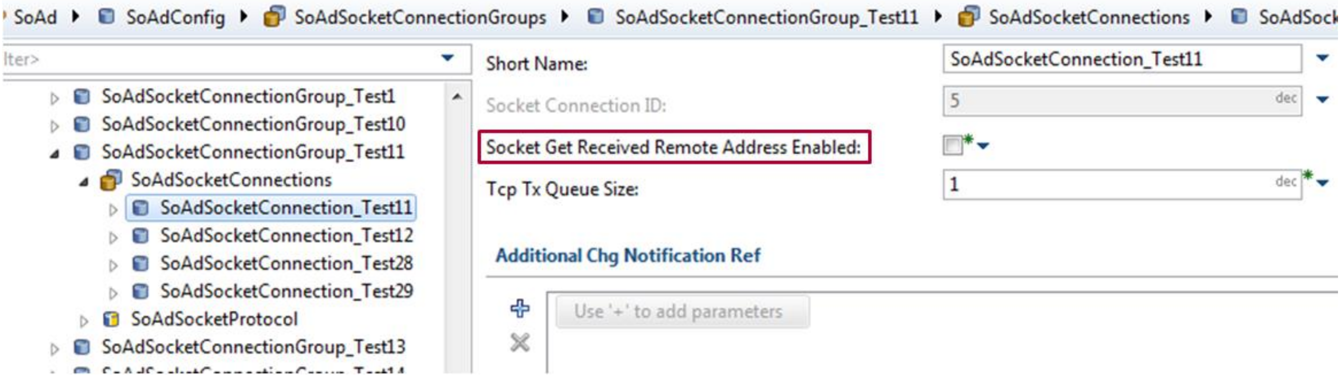
Figure 5-11 UDP Immediate IF TxConfirmation

According to [1] TxConfirmation for UDP socket connections with IF API is called within main function. It may be possible that new data are received while sent data are not confirmed yet. This may lead to incorrect behavior if upper layer is based on request and response behavior. Immediate TxConfirmaion can prevent reception before confirmation.

This feature is implemented for one `PduRoute/PduRouteDestination` per `SoAdSocketConnection` and one `SoAdSocketConnection` per `SoAdSocketConnectionGroup` only.

5.3.4.3 SoAd_GetRcvRemoteAddr()

Via this API it is possible for the upper layer to retrieve the remote address of the last received message on a socket connection. To enable this feature enable the parameter marked in Figure 5-12.


SoAd ▸ SoAdConfig ▸ SoAdSocketConnectionGroups ▸ SoAdSocketConnectionGroup_Test11 ▸ SoAdSocketConnections ▸ SoAdSock
lter>
SoAdSocketConnectionGroup_Test1
SoAdSocketConnectionGroup_Test10
SoAdSocketConnectionGroup_Test11
SoAdSocketConnections
SoAdSocketConnection_Test11
SoAdSocketConnection_Test12
SoAdSocketConnection_Test28
SoAdSocketConnection_Test29
SoAdSocketProtocol
SoAdSocketConnectionGroup_Test13
Additional Chg Notification Ref
Use '+' to add parameters

Short Name: SoAdSocketConnection_Test11

Socket Connection ID: 5 dec

Socket Get Received Remote Address Enabled: □

Tcp Tx Queue Size: 1 dec

Figure 5-12 SoAd_GetRcvRemoteAddr

5.3.4.4 Additional change notifications

The Socket Adaptor supports several use-cases to configure additional notification recipients on change of the socket connection mode and the local IP address assignment in case the corresponding change notification is enabled.

- Enable `<Up>_SoConModeChg()` for all socket connections of a socket connection group for a specific `SoAdBswModule:`

  It is required to configure `SoAdSocketSoConModeChgNotifUpperLayerRef` on a `SoAdSocketConnectionGroup` to reference a `SoAdBswModule` which is not referenced by a corresponding `SoAdPduRoute` or `SoAdSocketRouteDest.`

- Enable `<Up>_SoConModeChg()` and/or `<Up>_LocalIpAddrAssignmentChg()` for a particular socket connection for a specific `SoAdBswModule:`

  It is required to configure `SoAdAdditionalSoConModeChgRef` on a `SoAdSocketConnection` to reference a `SoAdBswModule` which is not referenced by a corresponding `SoAdPduRoute` or `SoAdSocketRouteDest.`

- Enable `<Up>_SoConModeChg()` and/or `<Up>_LocalIpAddrAssignmentChg()` for a specific socket connection for a custom user/module:

  It is possible to configure additional callbacks to notify any module on change of the socket connection mode or local IP address assignment correspondingly.

For `<Up>_SoConModeChg()`, it is required to configure `SoAdAdditionalSoConModeChgRef` on a `SoAdSocketConnection` to reference a `SoAd/SoAdGeneral/SoAdAdditionalSoConModeChgCallback/SoAdAdditionalSoConModeChg`.

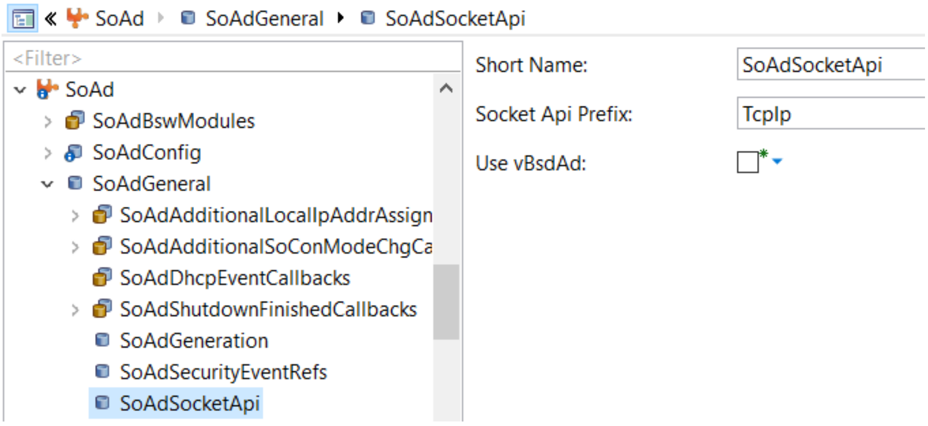
For `<Up>_LocalIpAddrAssignmentChg()`, it is required to configure `SoAdAdditionalLocalIpAddrAssignmentChgRef` on a `SoAdSocketConnection` to reference a `SoAd/SoAdGeneral/SoAdAdditionalLocalIpAddrAssignmentChgCallback/SoAdAdditionalLocalIpAddrAssignmentChg`.

5.3.4.5 Socket API

Since Socket Adaptor version 17.00.00, the configuration of a BSD Socket API inside the Socket Adaptor module is no longer available. However, it is supported to use SoAd with different Socket APIs. By default, the AUTOSAR TcpIp stack is used.

The Socket API is configured by the parameter `SoAdSocketApiPrefix` inside the container `SoAdSocketApi` as shown in Figure 5-13. By default, the Socket API Prefix is set to TcpIp for usage of the AUTOSAR TcpIp module. The parameter defines the function name prefix for the Socket API. Table 4-41 lists the <TcpIp> services used by Socket Adaptor. If e.g. the prefix is set to vBsdTcpIp, SoAd calls the function `vBsdTcpIp_Close()` to close a socket. Therefore, it must be ensured that all these services are available in the configured Socket API. Additionally, the following include-files depending on the Socket API Prefix are included in the generated files and must be available:

1. <TcpIp>.h
2. <TcpIp>_Types.h


« SoAd › SoAdGeneral › SoAdSocketApi
<Filter>
SoAd
SoAdBswModules
SoAdConfig
SoAdGeneral
SoAdAdditionalLocalIpAddrAssign
SoAdAdditionalSoConModeChgCa
SoAdDhcpEventCallbacks
SoAdShutdownFinishedCallbacks
SoAdGeneration
SoAdSecurityEventRefs
SoAdSocketApi

Short Name: SoAdSocketApi

Socket Api Prefix: TcpIp

Use vBsdAd: □

Figure 5-13 Configure Socket API

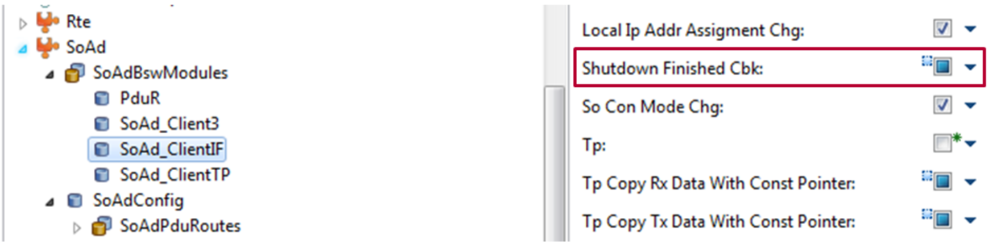
In addition to the `SoAdSocketApiPrefix` parameter, the boolean parameter `SoAdVBsdAdUsed` exists (also shown in Figure 5-13). This parameter must be enabled when Socket Adaptor is used together with vBsdAd component as not all features are supported when vBsdAd is used as lower layer. Please refer to the description of the parameter for an enumeration of the affected features.

5.3.4.6 Shutdown mechanism

Service `SoAd_Shutdown()` initiates a shutdown of Socket Adaptor module. Pending transmissions and receptions will be continued, new transmission and reception requests will be rejected and Socket Adaptor is set into a shutdown state afterwards. A call to `SoAd_Init()` is required to reinitialize Socket Adaptor again.

This feature is enabled always and can be used to shutdown Socket Adaptor orderly (e.g. in case of flashing the ECU).

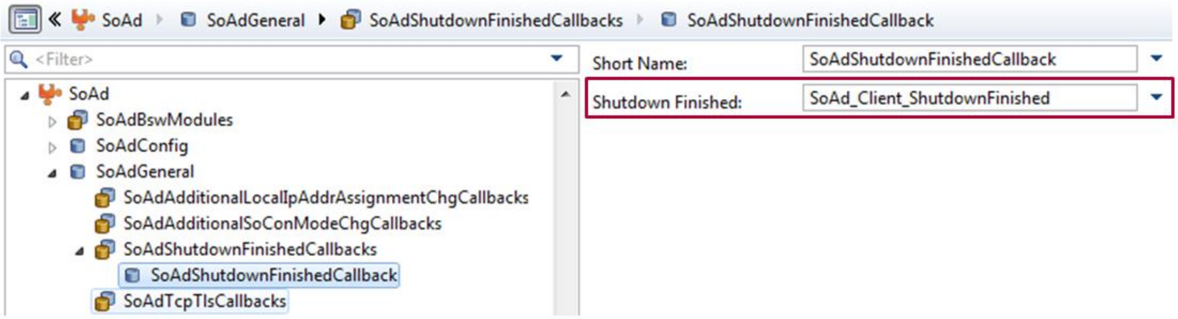
It is possible to configure an optional callback which is called when shutdown is finished. This callback can be configured as described in Figure 5-14 for a specific upper layer or in Figure 5-15 for any other module (please consider Figure 5-16 to include additional header files).



Rte
SoAd
SoAdBswModules
PduR
SoAd_Client3
SoAd_ClientIF
SoAd_ClientTP
SoAdConfig
SoAdPduRoutes

Local Ip Addr Assigment Chg: ☑
Shutdown Finished Cbk:
So Con Mode Chg: ☑
Tp: □
Tp Copy Rx Data With Const Pointer:
Tp Copy Tx Data With Const Pointer:

Figure 5-14 Enable upper layer specific <User>_ShutdownFinished()



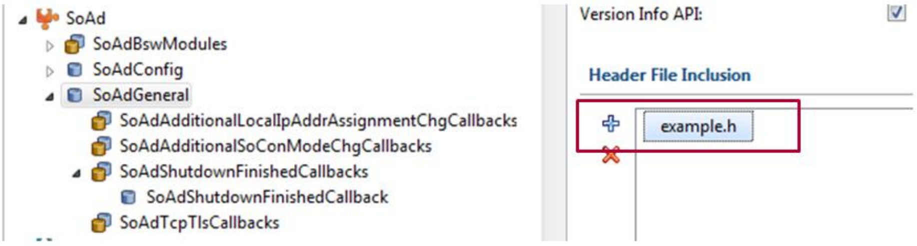
SoAd › SoAdGeneral › SoAdShutdownFinishedCallbacks › SoAdShutdownFinishedCallback

<Filter>

SoAd
SoAdBswModules
SoAdConfig
SoAdGeneral
SoAdAdditionalLocalIpAddrAssignmentChgCallbacks
SoAdAdditionalSoConModeChgCallbacks
SoAdShutdownFinishedCallbacks
SoAdShutdownFinishedCallback
SoAdTcpTlsCallbacks

Short Name: SoAdShutdownFinishedCallback
Shutdown Finished: SoAd_Client_ShutdownFinished

Figure 5-15 Configure <User>_ShutdownFinished() for any module



SoAd
SoAdBswModules
SoAdConfig
SoAdGeneral
SoAdAdditionalLocalIpAddrAssignmentChgCallbacks
SoAdAdditionalSoConModeChgCallbacks
SoAdShutdownFinishedCallbacks
SoAdShutdownFinishedCallback
SoAdTcpTlsCallbacks

Version Info API: ☑

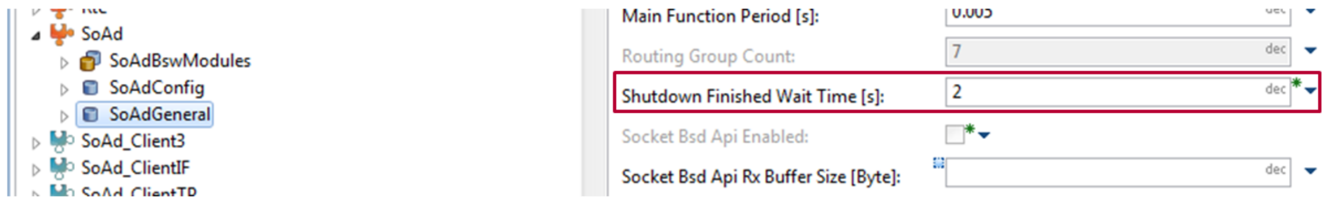
Header File Inclusion

example.h

Figure 5-16 Additional header file inclusion

It is also possible to poll module state via multiple calls to `SoAd_Shutdown()`.

If a pending transmission or reception cannot be finished or sockets cannot be closed since tester does not acknowledge closing (i.e. no "FIN" flag sent) a timeout is configured to shutdown module immediately after timeout (Figure 5-17) expired. This value must be configured to at least twice the main function period. The before last main function cycle takes care of closing the sockets while the last one finishes the shutdown by setting the state and by notifying the user.



Main Function Period [s]: 0.005

Routing Group Count: 7

Shutdown Finished Wait Time [s]: 2

Socket Bsd Api Enabled: □

Socket Bsd Api Rx Buffer Size [Byte]:

Figure 5-17 Shutdown finished timeout

5.3.4.7 MainFunction splitting

Depending on upper layer API (TP/IF) and protocol (TCP/UDP) transmission and reception are handled in MainFunction context of Socket Adaptor (refer to [1]). Also other modules of the communication stack use MainFunctions for transmission and reception (e.g. TcpIp). If there is only one MainFunction per module it has to be decided if order of calls to the MainFunctions is optimized for transmission or reception path.

Order for reception optimization:

```
TcpIp_MainFunction() -> SoAd_MainFunction()
```

Order for transmission optimization:

```
SoAd_MainFunction() -> TcpIp_MainFunction()
```

Vector supports a spitting into Rx MainFunction and Tx MainFunction to optimize transmission and reception path parallel.

Order in combination with splitting of MainFunction of TcpIp module:

```
TcpIp_MainFunctionRx() -> SoAd_MainfunctionRx() ->
SoAd_MainfunctionTx() -> TcpIp_MainFunctionTx()
```

Additionally `SoAd_MainFunctionState()` is available to handle the module states.

To enable the feature set the following parameter to true:

```
SoAd/SoAdGeneral/SoAdMainFunctionSplitEnabled
```

Caution
If splitting is enabled `SoAd_MainFunction()` must not be called anymore.

5.3.4.8 Optimized TP transmission

Socket Adaptor according to AUTOSAR can handle maximum one TP transmission per MainFunction on a PDU. But some upper layers need to send data more often.

Therefore, an optimized TP transmission is supported. If this feature is enabled the entire TP transmission can be handled in transmission context as described in Figure 5-18.

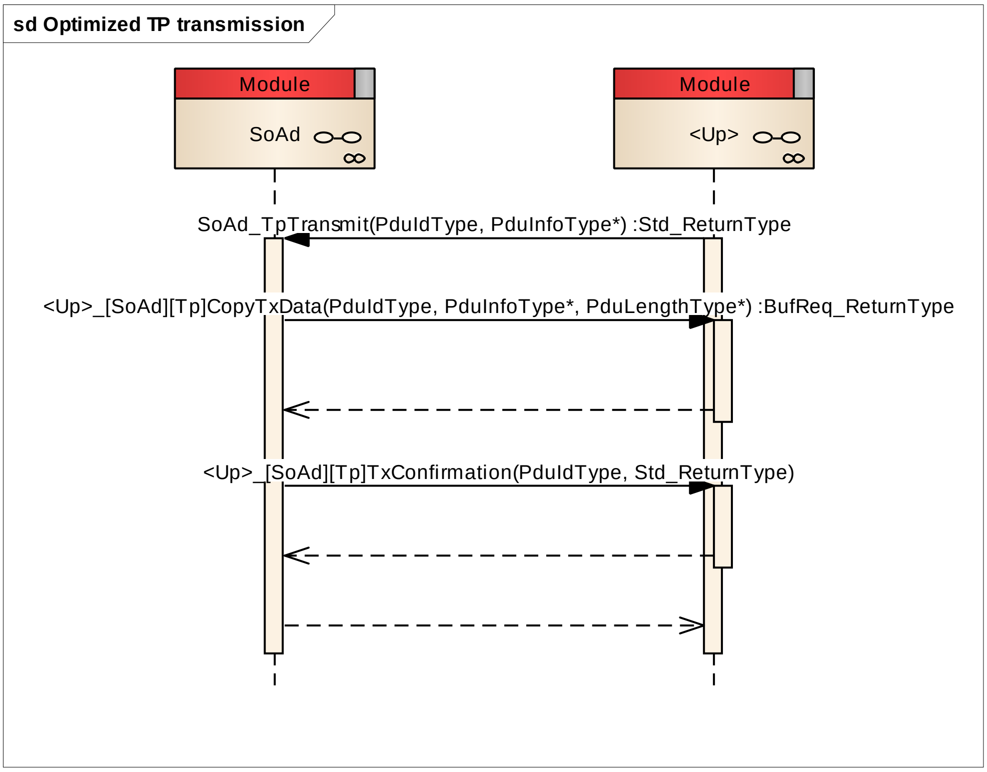
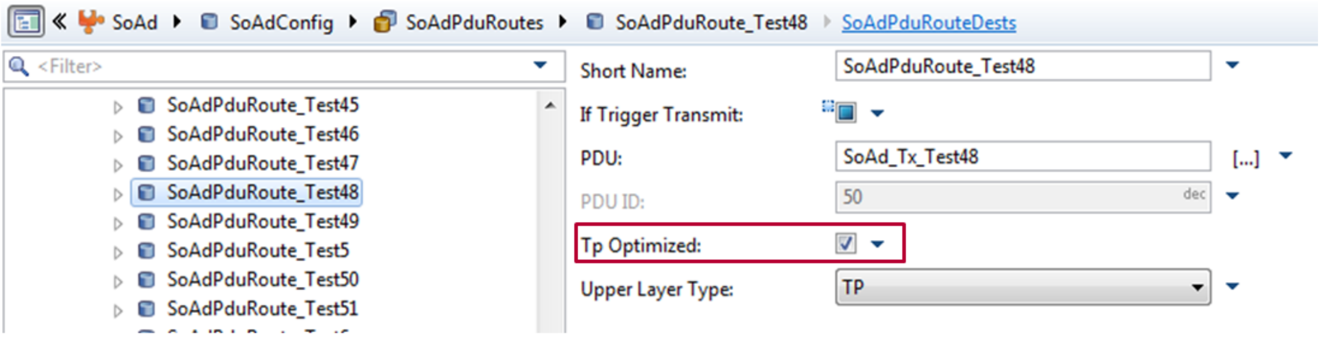

sd Optimized TP transmission
Module
SoAd
Module
<Up>
SoAd_TpTransmit(PduIdType, PduInfoType*) :Std_ReturnType
<Up>_[SoAd][Tp]CopyTxData(PduIdType, PduInfoType*, PduLengthType*) :BufReq_ReturnType
<Up>_[SoAd][Tp]TxConfirmation(PduIdType, Std_ReturnType)

Figure 5-18 Optimized TP transmission sequence

This feature can be enabled and disabled for each TP `SoAdPduRoute` separately (Figure 5-19).



Short Name: SoAdPduRoute_Test48

If Trigger Transmit: □

PDU: SoAd_Tx_Test48

PDU ID: 50

Tp Optimized: ☑

Upper Layer Type: TP

Figure 5-19 Optimized TP transmission configuration

Caution
Optimized TP transmission is not a standard MICROSAR Classic feature. Ensure that the feature is supported by the components related to the `SoAdPduRoute`.

Note
It is highly recommended to enable `SoAdSocketTcpImmediateTpTxConfirmation` if feature is used on TCP sockets since TP transmission is finished on reception of the TCP ACK otherwise and not in context of TP transmission request.

Note
Please consider `SoAdTcpTxQueueSize` since even if TP transmission is finished corresponding queue element is not released until reception of TCP ACK.

5.3.4.9 Trigger Transmit extensions

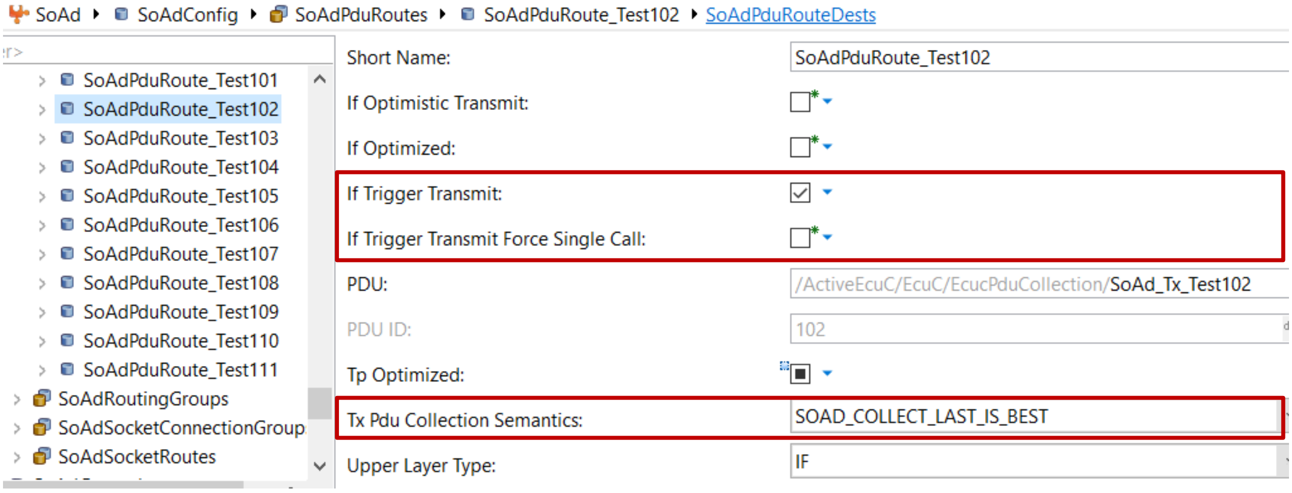
According to [1] Trigger Transmit can be enabled per `SoAdBswModule` by the parameter `SoAdIfTriggerTransmit`. With MICROSAR Classic, it is possible to enable and disable this feature per `SoAdPduRoute` by configuring `SoAdTxIfTriggerTransmit` (refer to Figure 5-20). Please note that there is a dependency between the AUTOSAR parameter `SoAdTxPduCollectionSemantics` (also shown in Figure 5-20) and the MICROSAR Classic parameter `SoAdTxIfTriggerTransmit`. The configured values of these two parameters must match as follows:

1. When `SoAdTxIfTriggerTransmit` is enabled while `SoAdTxIfTriggerTransmitForceSingleCall` is disabled and nPdu transmission is used, it is expected that `SoAdTxPduCollectionSemantics` is set to `SOAD_COLLECT_LAST_IS_BEST`.
2. Otherwise, it is expected that `SoAdTxPduCollectionSemantics` is set to `SOAD_COLLECT_QUEUED`.

Caution
`SoAd_IfTransmit` has to be called with valid length (e.g. length of PDU instead of zero) since length is needed to call TcpIp transmission service before Trigger Transmit callback is called.

For some reasons length of PDU maybe unknown on call to `SoAd_IfTransmit`. In this case enable `SoAdTxDynamicLengthEnabled` in configuration tool to switch to a Vector specific transmission API between SoAd and TcpIp. With this adapted API it is possible to request a specific length but copy less data. So if PDU length is unknown maximum PDU length can be used in `SoAd_IfTransmit` and smaller length can be copied in Trigger Transmit callback.

Caution
The adapted API has to be supported by the TcpIp module.



SoAd ▸ SoAdConfig ▸ SoAdPduRoutes ▸ SoAdPduRoute_Test102 ▸ SoAdPduRouteDests

Short Name: SoAdPduRoute_Test102

If Optimistic Transmit: □

If Optimized: □

If Trigger Transmit: ☑

If Trigger Transmit Force Single Call: □

PDU: /ActiveEcuC/EcuC/EcucPduCollection/SoAd_Tx_Test102

PDU ID: 102

Tp Optimized: ■

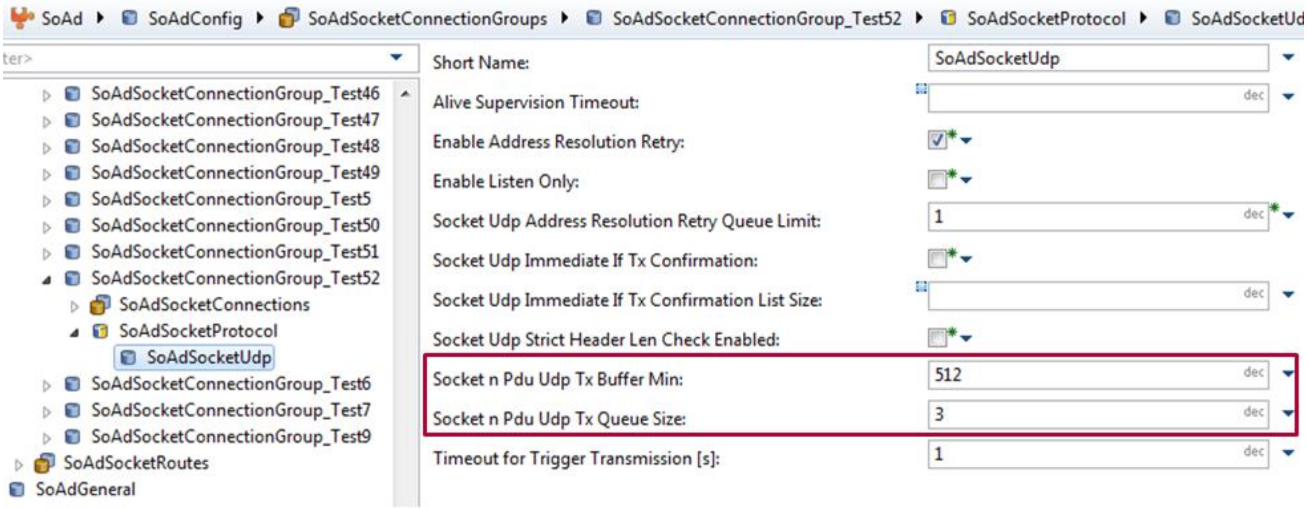
Tx Pdu Collection Semantics: SOAD_COLLECT_LAST_IS_BEST

Upper Layer Type: IF

Figure 5-20 Trigger Transmit parameters on SoAdPduRoute configuration

5.3.4.10 Queue size as trigger condition for nPdu feature

AUTOSAR specifies that it shall be possible to store transmission requests if the nPdu feature is used and `SoAdTxPduCollectionSemantics` is configured to `SOAD_COLLECT_LAST_IS_BEST`. This means that a transmission request for a PDU is stored in a queue instead of the entire PDU in a buffer and if the trigger condition is fulfilled for the nPdu PDU data are retrieved via the Trigger Transmit API (refer to chapter 5.3.4.9). The PDU is stored once in the queue even if PDU transmission is triggered multiple times. Further transmission requests to this PDU will update the length information stored in the queue. To make the maximum amount of stored PDUs configurable, the parameter `SoAdSocketnPduUdpTxQueueSize` is introduced to configure the queue size per socket connection. If a new element exceeds the queue size the nPdu is sent like specified for exceeding the nPduUdpTxBuffer. So, if the queueing mechanism is used the parameters `SoAdSocketnPduUdpTxBufferMin` and `SoAdSocketnPduUdpTxQueueSize` must be considered as trigger conditions (refer to Figure 5-21) but `SoAdSocketnPduUdpTxBufferMin` is a transmission trigger condition only and no buffer is generated.



SoAd ▸ SoAdConfig ▸ SoAdSocketConnectionGroups ▸ SoAdSocketConnectionGroup_Test52 ▸ SoAdSocketProtocol ▸ SoAdSocketUd

ter>

- SoAdSocketConnectionGroup_Test46
- SoAdSocketConnectionGroup_Test47
- SoAdSocketConnectionGroup_Test48
- SoAdSocketConnectionGroup_Test49
- SoAdSocketConnectionGroup_Test5
- SoAdSocketConnectionGroup_Test50
- SoAdSocketConnectionGroup_Test51
- SoAdSocketConnectionGroup_Test52
  - SoAdSocketConnections
  - SoAdSocketProtocol
    - SoAdSocketUdp
- SoAdSocketConnectionGroup_Test6
- SoAdSocketConnectionGroup_Test7
- SoAdSocketConnectionGroup_Test9
- SoAdSocketRoutes
- SoAdGeneral

Short Name:	SoAdSocketUdp
Alive Supervision Timeout:	
Enable Address Resolution Retry:	☑
Enable Listen Only:	□
Socket Udp Address Resolution Retry Queue Limit:	1
Socket Udp Immediate If Tx Confirmation:	□
Socket Udp Immediate If Tx Confirmation List Size:	
Socket Udp Strict Header Len Check Enabled:	□
Socket n Pdu Udp Tx Buffer Min:	512
Socket n Pdu Udp Tx Queue Size:	3
Timeout for Trigger Transmission [s]:	1

Figure 5-21 NPduUdpTxQueue configuration

Note
It is possible to have an nPduUdpTxBuffer and an nPduUdpTxQueue on one socket connection. Refer to chapter 5.3.4.11 for more details.

5.3.4.11 Mixed semantic support for nPdu feature

Two different PDU collection semantics (`SoAdTxPduCollectionSemantics)` exist for nPdu transmission. In case of `SOAD_COLLECT_QUEUED`, all PDUs are stored in an nPduUdpTxBuffer and transmitted when the buffer is full, or any other transmission trigger applies. For `SOAD_COLLECT_LAST_IS_BEST` the transmission requests are stored in a queue (nPduUdpTxQueue, refer to chapter 5.3.4.10) instead of the entire PDU in a buffer and if the trigger condition is fulfilled the PDU data are retrieved via the Trigger Transmit API (refer to chapter 5.3.4.9).

AUTOSAR does not allow configurations in which the setting of `SoAdTxPduCollectionSemantics` on the PDUs which are assigned to a socket connection is mixed. Such a mixed semantic is supported by the MICROSAR Classic Socket Adaptor so that it is possible to have an nPduUdpTxBuffer and an nPduUdpTxQueue on one socket connection. In this case, PDUs in the nPduUdpTxBuffer are also represented by an element in the nPduUdpTxQueue.

5.3.4.12 Event Queues and Timeout Lists

5.3.4.12.1 Event Queues

Internally Socket Adaptor uses several event queues to store different events (e.g. socket connection states) to handle them in main function context. Each main function only handles the event queue entries of the corresponding SoAd instance (refer to 5.3.4.18 for more details).

It is possible to configure limitations for the event queues to reduce runtime in main function if many events occur at the same time. The limitation restricts the maximum number of executed events in one main function cycle. No events get lost if the limit is reached but they are executed in one of the next main function cycles (Figure 5-22).

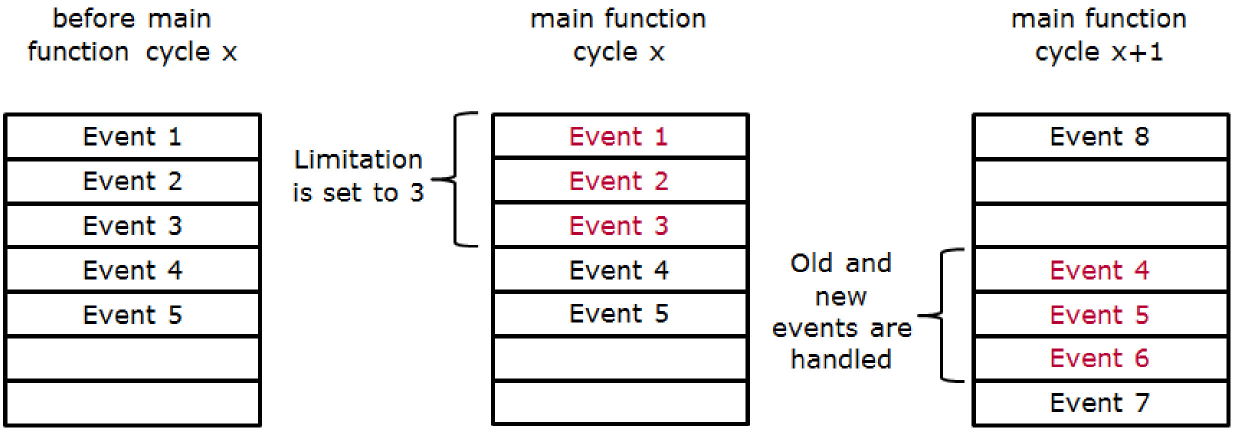
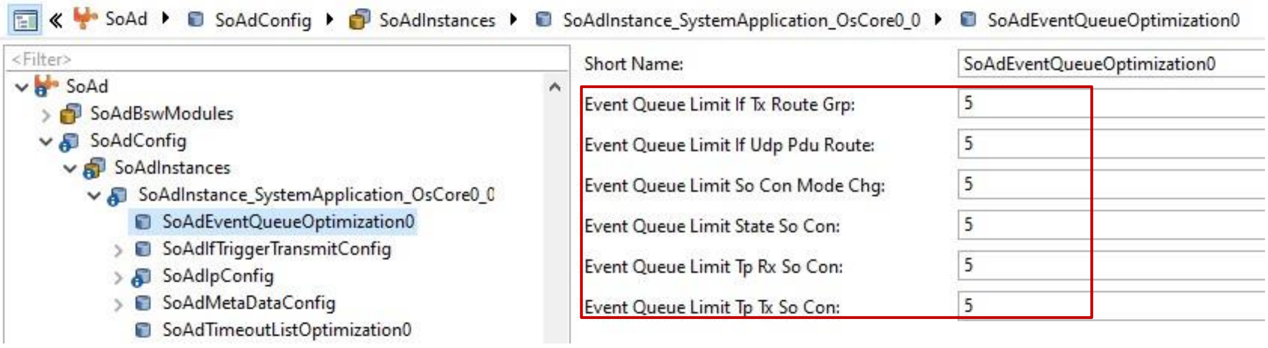


Figure 5-22 Event Queue limitation principle

The configuration of event queue parameters is required to be done per Socket Adaptor instance. To configure the event queue limitation create the corresponding configuration container and set the event queue limit parameters (Figure 5-23). If a container or parameter does not exist limitation is disabled (default) and all possible events are handled within one main function.


« SoAd ▸ SoAdConfig ▸ SoAdInstances ▸ SoAdInstance_SystemApplication_OsCore0_0 ▸ SoAdEventQueueOptimization0
<Filter>
SoAd
SoAdBswModules
SoAdConfig
SoAdInstances
SoAdInstance_SystemApplication_OsCore0_0
SoAdEventQueueOptimization0
SoAdIfTriggerTransmitConfig
SoAdIpConfig
SoAdMetaDataConfig
SoAdTimeoutListOptimization0

Short Name: SoAdEventQueueOptimization0

Event Queue Limit If Tx Route Grp: 5

Event Queue Limit If Udp Pdu Route: 5

Event Queue Limit So Con Mode Chg: 5

Event Queue Limit State So Con: 5

Event Queue Limit Tp Rx So Con: 5

Event Queue Limit Tp Tx So Con: 5

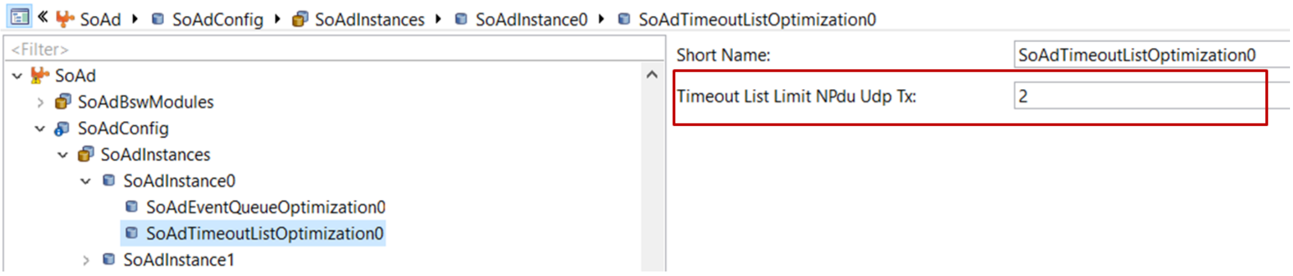
Figure 5-23 Event Queue limitation configuration

5.3.4.12.2 Timeout Lists

Beside event queues, Socket Adaptor uses timeout lists to handle timeouts in main function (e.g. UDP alive supervision timeout). The timeout list configuration and main function handling is also done per SoAd instance.

It is also possible to configure limitations for the timeout lists. The limitation limits the size of the corresponding timeout list itself. So if limitation is reached a new timeout cannot be added to list and for example in case of Trigger Timeout for nPdus the corresponding transmission request is rejected.

To configure the timeout list limitation create the corresponding configuration container and set the timeout list limit (Figure 5-24). If container or parameter does not exist limitation is disabled (default) and all possible timeouts can be handled.


SoAd ▸ SoAdConfig ▸ SoAdInstances ▸ SoAdInstance0 ▸ SoAdTimeoutListOptimization0
<Filter>
SoAd
SoAdBswModules
SoAdConfig
SoAdInstances
SoAdInstance0
SoAdEventQueueOptimization0
SoAdTimeoutListOptimization0
SoAdInstance1

Short Name: SoAdTimeoutListOptimization0

Timeout List Limit NPdu Udp Tx: 2

Figure 5-24 Timeout List limitation configuration

5.3.4.13 PDU reception verification

For TCP socket connections using the PDU Header option over the TP-API Socket Adaptor supports a PDU reception verification callback which can be used to filter a received PDU according to the following parameters:

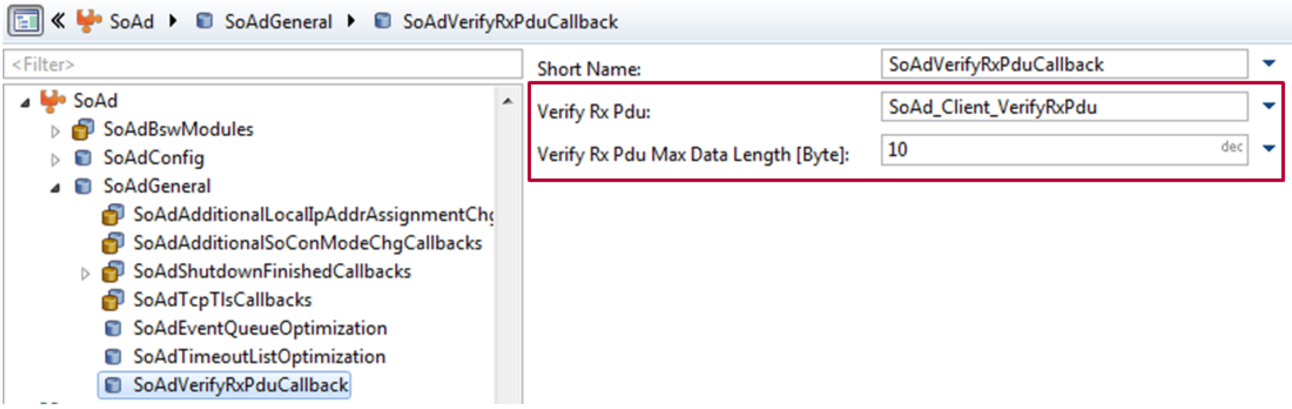
1. Local IP address and port
2. Remote IP address and port
3. PDU Header ID
4. PDU data

This feature can be used to implement a firewall on Socket Adaptor level.

In case callback `<Up_VerifyRxPdu>` is successful Socket Adaptor forwards the PDU as configured. In case callback fails Socket Adaptor drops the PDU silently and continues with the reception of PDUs received afterwards.

Figure 5-25 shows how to configure the name of the callback and the maximum amount of PDU data which are forwarded via the callback.

Figure 5-26 shows how to enable the call of the callback for a specific socket connection. If disabled on a socket connection the callback won't be called.

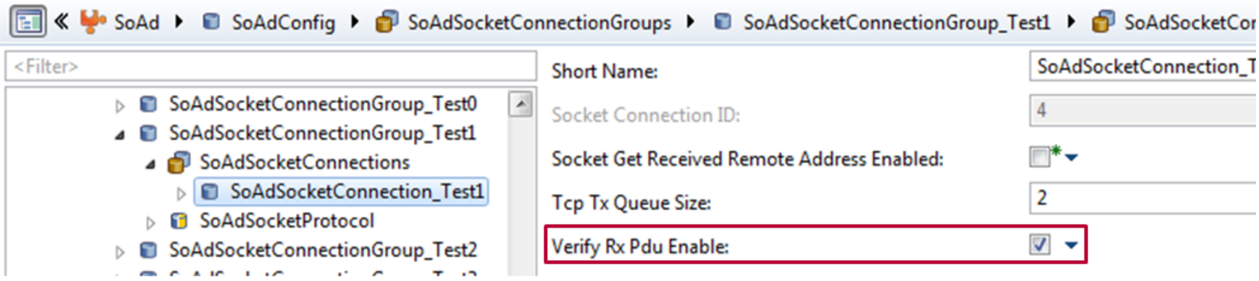

SoAd ▸ SoAdGeneral ▸ SoAdVerifyRxPduCallback
<Filter>
SoAd
SoAdBswModules
SoAdConfig
SoAdGeneral
SoAdAdditionalLocalIpAddrAssignmentChg
SoAdAdditionalSoConModeChgCallbacks
SoAdShutdownFinishedCallbacks
SoAdTcpTlsCallbacks
SoAdEventQueueOptimization
SoAdTimeoutListOptimization
SoAdVerifyRxPduCallback

Short Name: SoAdVerifyRxPduCallback

Verify Rx Pdu: SoAd_Client_VerifyRxPdu

Verify Rx Pdu Max Data Length [Byte]: 10 dec

Figure 5-25 PDU reception verification callback configuration


« SoAd ▸ SoAdConfig ▸ SoAdSocketConnectionGroups ▸ SoAdSocketConnectionGroup_Test1 ▸ SoAdSocketCor
<Filter>
SoAdSocketConnectionGroup_Test0
SoAdSocketConnectionGroup_Test1
SoAdSocketConnections
SoAdSocketConnection_Test1
SoAdSocketProtocol
SoAdSocketConnectionGroup_Test2

Short Name: SoAdSocketConnection_T

Socket Connection ID: 4

Socket Get Received Remote Address Enabled: □

Tcp Tx Queue Size: 2

Verify Rx Pdu Enable: ☑

Figure 5-26 PDU reception verification socket connection configuration

5.3.4.14 “Transmission on specific socket connection” and “Forward socket connection on reception”

Both “Transmission on specific socket connection” and “Forward socket connection on reception” are features based on PDU meta data.

“Forward socket connection on reception” describes the mechanism to append the socket connection index as meta data after the payload data of a received PDU. This feature can be used to get the socket connection index on which the PDU has been received.

“Transmission on specific socket connection” describes the mechanism to extract the socket connection index from meta data of a PDU which shall be transmitted. Socket Adaptor transmits the PDU over the corresponding socket connection. This feature can be used to restrict the PDU transmission to a specific socket connection in case a PDU fan-out (see [1]) is configured. This feature is available for IF-PDUs on UDP and TCP socket connections and it can be used in regular transmission procedures as well as in combination with the trigger transmit-feature.

Both features can be used to configure an optimization for the modeling of Client/Server Calls (i.e. methods). For more details please see “RfC 64808” in the AUTOSAR Bugzilla.

To enable the usage of the meta data create the following parameter for the corresponding PDU in the EcuC module:

```
EcuC/EcucPduCollection/Pdu/MetaDataLength
```

Set the value to 2 since 2 bytes is the size of the socket connection index.

Note

Meta data is used in several MICROSAR Classic components. The handling of meta data is implemented differently and depends on the component-specific use-case. In the Socket Adaptor, the meta data is intended to be used for PduR-API-Forwarding to RTE and SomeIp and for communication via SomeIpTp. Other use-cases like multi-partition or gateway routing are not supported explicitly.

5.3.4.15 Get DHCP option events

There are no indications as to when a DHCP option can be read from a received DHCP message or when a DHCP option can be written into a DHCP message. For this reason, callbacks can be configured for the user to be notified of DHCP message reception and DHCP message transmission events.

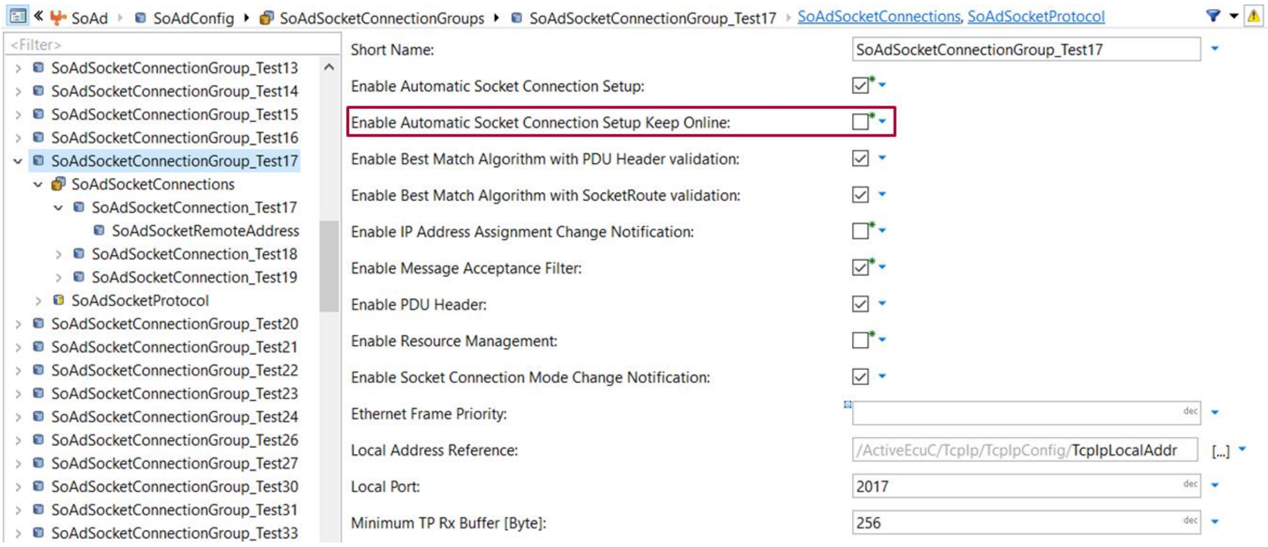
They can be configured using the parameter:

`SoAd/SoAdGeneral/SoAdDhcpEventCallback`

5.3.4.16 Keep Socket Connection online after transmission

According to AUTOSAR, a socket connection can be opened on UDP reception (SWS_SoAd_00592) or on TCP connection establishment (SWS_SoAd_00594). This applies if the configured remote address contains wildcards. After transmission of a PDU, the socket connection shall be closed (SWS_SoAd_00582 for UDP and SWS_SoAd_00644 for TCP).

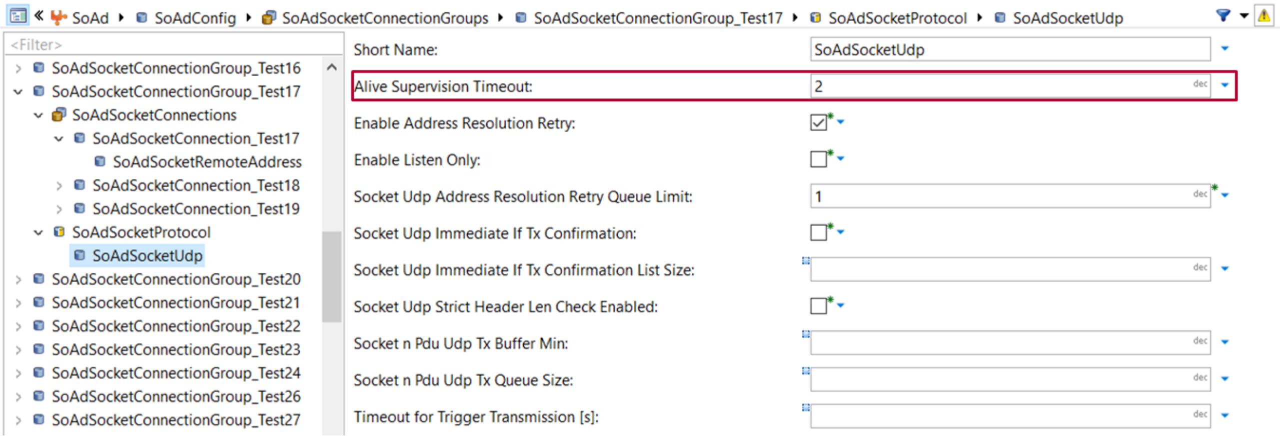
In some applications it might make sense to keep a wildcard socket connection online after transmission to prevent a repeated opening and closing. This can be configured by enabling the corresponding parameter `SoAdSocketAutomaticSoConSetupKeepOnline` shown in Figure 5-27. The setting only applies if the automatic socket connection setup is enabled and if the socket connection group contains at least one remote address which is set to wildcard.


SoAd
SoAdConfig
SoAdSocketConnectionGroups
SoAdSocketConnectionGroup_Test17
SoAdSocketConnections, SoAdSocketProtocol
<Filter>
SoAdSocketConnectionGroup_Test13
SoAdSocketConnectionGroup_Test14
SoAdSocketConnectionGroup_Test15
SoAdSocketConnectionGroup_Test16
SoAdSocketConnectionGroup_Test17
SoAdSocketConnections
SoAdSocketConnection_Test17
SoAdSocketRemoteAddress
SoAdSocketConnection_Test18
SoAdSocketConnection_Test19
SoAdSocketProtocol
SoAdSocketConnectionGroup_Test20
SoAdSocketConnectionGroup_Test21
SoAdSocketConnectionGroup_Test22
SoAdSocketConnectionGroup_Test23
SoAdSocketConnectionGroup_Test24
SoAdSocketConnectionGroup_Test26
SoAdSocketConnectionGroup_Test27
SoAdSocketConnectionGroup_Test30
SoAdSocketConnectionGroup_Test31
SoAdSocketConnectionGroup_Test33

Short Name: SoAdSocketConnectionGroup_Test17
Enable Automatic Socket Connection Setup: ☑
Enable Automatic Socket Connection Setup Keep Online: □
Enable Best Match Algorithm with PDU Header validation: ☑
Enable Best Match Algorithm with SocketRoute validation: ☑
Enable IP Address Assignment Change Notification: □
Enable Message Acceptance Filter: ☑
Enable PDU Header: ☑
Enable Resource Management: □
Enable Socket Connection Mode Change Notification: ☑
Ethernet Frame Priority:
Local Address Reference: /ActiveEcuC/TcpIp/TcpIpConfig/TcpIpLocalAddr [...]
Local Port: 2017
Minimum TP Rx Buffer [Byte]: 256

Figure 5-27 Keep Socket Connection online configuration

When this feature is enabled, it is recommended to use a timeout for automatic closing of the socket connections. For UDP, the `SoAdSocketUdpAliveSupervisionTimeout` can be used, which is shown in Figure 5-28. In case of a TCP socket connection it is required to enable `SoAdSocketTcpKeepAlive` and to set `SoAdSocketTcpKeepAliveProbesMax` to “0”, which is both configured in Figure 5-29. `SoAdSocketTcpKeepAliveTime` now defines the timeout value.

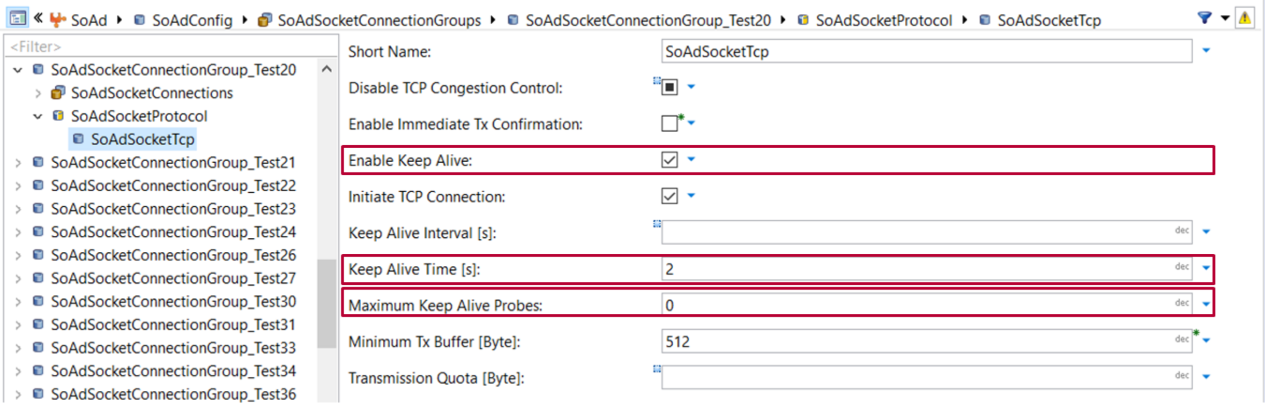


SoAd ▸ SoAdConfig ▸ SoAdSocketConnectionGroups ▸ SoAdSocketConnectionGroup_Test17 ▸ SoAdSocketProtocol ▸ SoAdSocketUdp

<Filter>
- SoAdSocketConnectionGroup_Test16
- SoAdSocketConnectionGroup_Test17
  - SoAdSocketConnections
    - SoAdSocketConnection_Test17
      - SoAdSocketRemoteAddress
    - SoAdSocketConnection_Test18
    - SoAdSocketConnection_Test19
  - SoAdSocketProtocol
    - SoAdSocketUdp
- SoAdSocketConnectionGroup_Test20
- SoAdSocketConnectionGroup_Test21
- SoAdSocketConnectionGroup_Test22
- SoAdSocketConnectionGroup_Test23
- SoAdSocketConnectionGroup_Test24
- SoAdSocketConnectionGroup_Test26
- SoAdSocketConnectionGroup_Test27

Short Name:	SoAdSocketUdp
Alive Supervision Timeout:	2
Enable Address Resolution Retry:	☑
Enable Listen Only:	□
Socket Udp Address Resolution Retry Queue Limit:	1
Socket Udp Immediate If Tx Confirmation:	□
Socket Udp Immediate If Tx Confirmation List Size:	
Socket Udp Strict Header Len Check Enabled:	□
Socket n Pdu Udp Tx Buffer Min:	
Socket n Pdu Udp Tx Queue Size:	
Timeout for Trigger Transmission [s]:	

Figure 5-28 Socket UDP Alive Supervision Timeout configuration



SoAd ▸ SoAdConfig ▸ SoAdSocketConnectionGroups ▸ SoAdSocketConnectionGroup_Test20 ▸ SoAdSocketProtocol ▸ SoAdSocketTcp

<Filter>
- SoAdSocketConnectionGroup_Test20
  - SoAdSocketConnections
  - SoAdSocketProtocol
    - SoAdSocketTcp
- SoAdSocketConnectionGroup_Test21
- SoAdSocketConnectionGroup_Test22
- SoAdSocketConnectionGroup_Test23
- SoAdSocketConnectionGroup_Test24
- SoAdSocketConnectionGroup_Test26
- SoAdSocketConnectionGroup_Test27
- SoAdSocketConnectionGroup_Test30
- SoAdSocketConnectionGroup_Test31
- SoAdSocketConnectionGroup_Test33
- SoAdSocketConnectionGroup_Test34
- SoAdSocketConnectionGroup_Test36

Short Name:	SoAdSocketTcp
Disable TCP Congestion Control:	■
Enable Immediate Tx Confirmation:	□
Enable Keep Alive:	☑
Initiate TCP Connection:	☑
Keep Alive Interval [s]:	
Keep Alive Time [s]:	2
Maximum Keep Alive Probes:	0
Minimum Tx Buffer [Byte]:	512
Transmission Quota [Byte]:	

Figure 5-29 Socket TCP Keep Alive Time configuration

5.3.4.17 Anomaly reporting

Socket Adaptor supports the reporting and counting of anomalies which can either be the dropping of PDUs, frames or connections due to unexpected values.

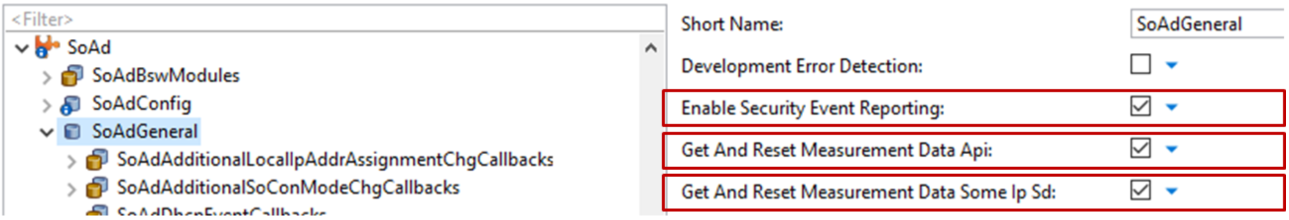
Anomalies are reported by Socket Adaptor in two different ways:

- Security events: The anomaly is reported on occurrence as a security event.
- Measurement data: Anomalies are counted on occurrence and reported to the user on request.

Security events are supported according to [6] and reported to the IdsM. Measurement data support some additional counters compared to [1]. Please refer to the `SoAd_MeasurementIdxType` in Table 4-1 for the available measurement counters.

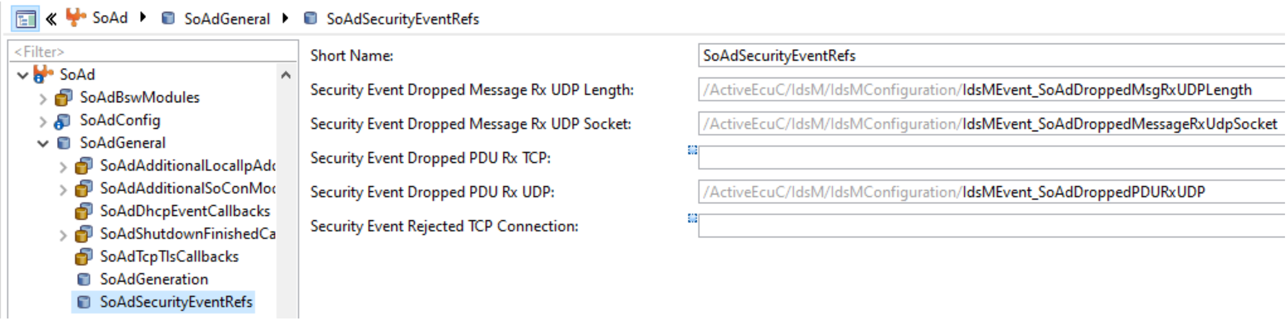
Security events are reported to IdsM when `SoAdEnableSecurityEventReporting` is enabled (refer to the first parameter in Figure 5-30) and `SoAdSecurityEventRefs` are configured as shown in Figure 5-31.

The counting of measurement data is enabled by configuration of `SoAdGetAndResetMeasurementDataApi`. For activation of SOME/IP and SD measurement data `SoAdGetAndResetMeasurementDataSomeIpSd` must be enabled additionally. Figure 5-30 shows the above-mentioned parameters.


<Filter>
SoAd
SoAdBswModules
SoAdConfig
SoAdGeneral
SoAdAdditionalLocalIpAddrAssignmentChgCallbacks
SoAdAdditionalSoConModeChgCallbacks

Short Name: SoAdGeneral
Development Error Detection: □
Enable Security Event Reporting: ☑
Get And Reset Measurement Data Api: ☑
Get And Reset Measurement Data Some Ip Sd: ☑

Figure 5-30 Anomaly reporting configuration



Short Name: SoAdSecurityEventRefs
Security Event Dropped Message Rx UDP Length: /ActiveEcuC/IdsM/IdsMConfiguration/IdsMEvent_SoAdDroppedMsgRxUDPLength
Security Event Dropped Message Rx UDP Socket: /ActiveEcuC/IdsM/IdsMConfiguration/IdsMEvent_SoAdDroppedMessageRxUdpSocket
Security Event Dropped PDU Rx TCP:
Security Event Dropped PDU Rx UDP: /ActiveEcuC/IdsM/IdsMConfiguration/IdsMEvent_SoAdDroppedPDURxUDP
Security Event Rejected TCP Connection:

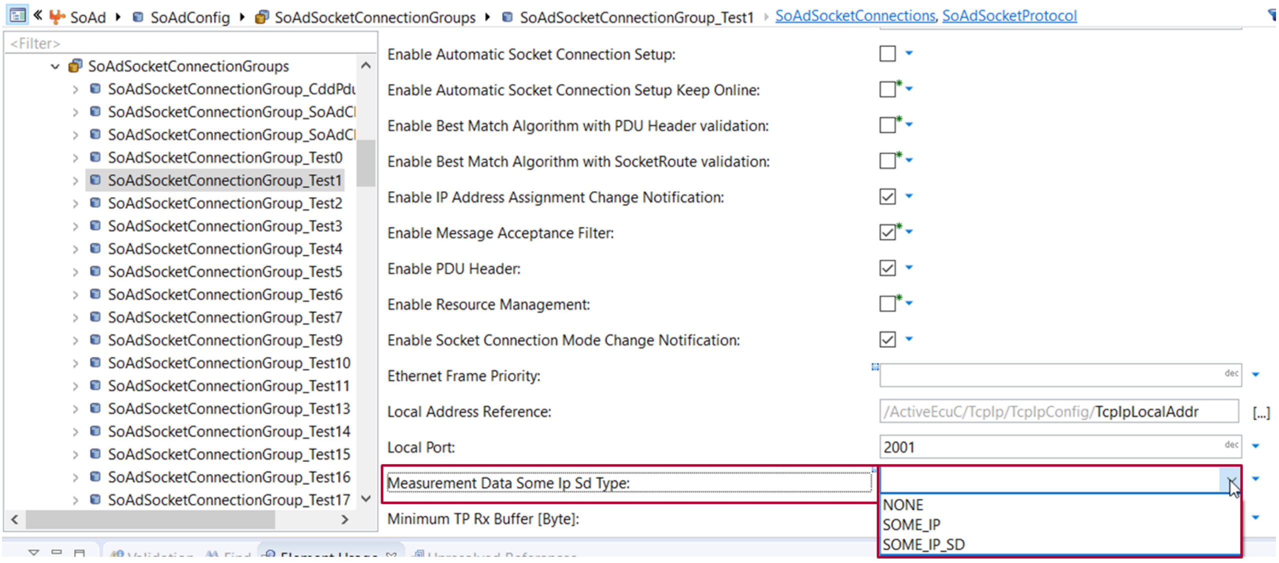
Figure 5-31 Security event references configuration

The SOME/IP and SD measurement data is only updated in case the optional parameter `SoAdMeasurementDataSomeIpSdType` is configured to either `SOME_IP` or `SOME_IP_SD` on the socket connection group (refer to Figure 5-32). This parameter must only be configured when the socket connection group or one of the socket connections of the socket connection group is used by a socket route having a PDU header ID configured. The first 2 bytes of the PDU header ID represent the service ID and the last 2 bytes represent the method ID. On reception of a PDU, the received PDU header ID is compared to the configured PDU header IDs of the socket connection which the PDU was received on. This is done by having a binary search on the service IDs. To make memory usage and runtime as efficient as possible, only the associated method IDs of matching service IDs are considered and there is no additional search on all method IDs. If no matching PDU header ID is found following options exist for incrementation of the counter values:

1. If no matching service ID was found, the service ID counter is incremented.
2. If a matching service ID was found but no matching PDU header ID, the method ID counter is incremented.

Whether the SOME/IP- or the SOME/IP SD counters are incremented depends on the configuration of the parameter on the socket connection group.

To get and reset the counter values, please refer to the API description of `SoAd_GetAndResetMeasurementData()` given in chapter 4.2.35.



SoAd ▸ SoAdConfig ▸ SoAdSocketConnectionGroups ▸ SoAdSocketConnectionGroup_Test1 ▸ SoAdSocketConnections, SoAdSocketProtocol

<Filter>

- SoAdSocketConnectionGroups
  - SoAdSocketConnectionGroup_CddPdu
  - SoAdSocketConnectionGroup_SoAdCl
  - SoAdSocketConnectionGroup_SoAdCl
  - SoAdSocketConnectionGroup_Test0
  - SoAdSocketConnectionGroup_Test1
  - SoAdSocketConnectionGroup_Test2
  - SoAdSocketConnectionGroup_Test3
  - SoAdSocketConnectionGroup_Test4
  - SoAdSocketConnectionGroup_Test5
  - SoAdSocketConnectionGroup_Test6
  - SoAdSocketConnectionGroup_Test7
  - SoAdSocketConnectionGroup_Test9
  - SoAdSocketConnectionGroup_Test10
  - SoAdSocketConnectionGroup_Test11
  - SoAdSocketConnectionGroup_Test13
  - SoAdSocketConnectionGroup_Test14
  - SoAdSocketConnectionGroup_Test15
  - SoAdSocketConnectionGroup_Test16
  - SoAdSocketConnectionGroup_Test17

Enable Automatic Socket Connection Setup: □
Enable Automatic Socket Connection Setup Keep Online: □
Enable Best Match Algorithm with PDU Header validation: □
Enable Best Match Algorithm with SocketRoute validation: □
Enable IP Address Assignment Change Notification: ☑
Enable Message Acceptance Filter: ☑
Enable PDU Header: ☑
Enable Resource Management: □
Enable Socket Connection Mode Change Notification: ☑
Ethernet Frame Priority:
Local Address Reference: /ActiveEcuC/TcpIp/TcpIpConfig/TcpIpLocalAddr
Local Port: 2001
Measurement Data Some Ip Sd Type:
Minimum TP Rx Buffer [Byte]:

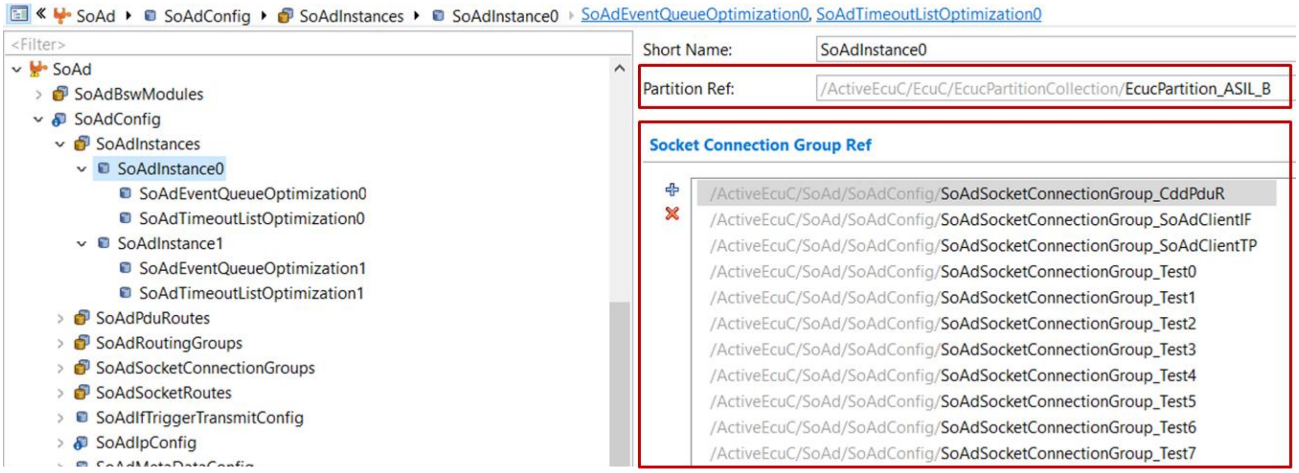
Figure 5-32 Measurement Data SOME/IP SD Type configuration

5.3.4.18 Support of Multi-partition

The Socket Adaptor supports the mapping of data paths (e.g. transmission or reception) to different partitions. Additionally, partition-independent functionality (e.g. shutdown mechanism) is supported on multi-partition systems.

To support multi-partition, SoAd defines instances which can be configured as shown in Figure 5-33. An instance is a set of parameters (e.g. event queue limitations) which applies to related data paths and corresponding configuration elements (e.g. socket connection group, PDU routes, ...).

Configuring more than one instance can be used without a partition mapping in a single-partition environment to e.g. logically separate different use-cases. Multi-partition is enabled by mapping the instances to different partitions (see the partition reference in Figure 5-33).



SoAd ▸ SoAdConfig ▸ SoAdInstances ▸ SoAdInstance0 ▸ SoAdEventQueueOptimization0, SoAdTimeoutListOptimization0

<Filter>

- SoAd
  - SoAdBswModules
  - SoAdConfig
    - SoAdInstances
      - SoAdInstance0
        - SoAdEventQueueOptimization0
        - SoAdTimeoutListOptimization0
      - SoAdInstance1
        - SoAdEventQueueOptimization1
        - SoAdTimeoutListOptimization1
    - SoAdPduRoutes
    - SoAdRoutingGroups
    - SoAdSocketConnectionGroups
    - SoAdSocketRoutes
    - SoAdIfTriggerTransmitConfig
    - SoAdIpConfig

Short Name: SoAdInstance0
Partition Ref: /ActiveEcuC/EcuC/EcucPartitionCollection/EcucPartition_ASIL_B

Socket Connection Group Ref

/ActiveEcuC/SoAd/SoAdConfig/SoAdSocketConnectionGroup_CddPduR
/ActiveEcuC/SoAd/SoAdConfig/SoAdSocketConnectionGroup_SoAdClientIF
/ActiveEcuC/SoAd/SoAdConfig/SoAdSocketConnectionGroup_SoAdClientTP
/ActiveEcuC/SoAd/SoAdConfig/SoAdSocketConnectionGroup_Test0
/ActiveEcuC/SoAd/SoAdConfig/SoAdSocketConnectionGroup_Test1
/ActiveEcuC/SoAd/SoAdConfig/SoAdSocketConnectionGroup_Test2
/ActiveEcuC/SoAd/SoAdConfig/SoAdSocketConnectionGroup_Test3
/ActiveEcuC/SoAd/SoAdConfig/SoAdSocketConnectionGroup_Test4
/ActiveEcuC/SoAd/SoAdConfig/SoAdSocketConnectionGroup_Test5
/ActiveEcuC/SoAd/SoAdConfig/SoAdSocketConnectionGroup_Test6
/ActiveEcuC/SoAd/SoAdConfig/SoAdSocketConnectionGroup_Test7

Figure 5-33 SoAd instance configuration

Configuration elements are mapped to an instance by adding the socket connection group references to the instance as shown in Figure 5-33. Each socket connection group as well as PDU routes (Tx data path) and socket routes (Rx data path) must only be referenced by one SoAd instance.

As already mentioned in chapter 5.3.4.12, event queue and timeout list parameters are defined instance-wise. Additionally, the available buffer pools (e.g. meta data or trigger transmit buffers) are configured per instance.

Moreover, the main function(s) (refer to chapter 2.5) exist per instance and each main function handles only the data of the corresponding instance. Refer to the chapters 4.2.36 - 4.2.39 for the naming convention of the main function(s) when more than one instance exists.

The shutdown feature (refer to 5.3.4.6) is used partition-independent. Whenever a shutdown is triggered all instances perform the steps to shut down the Socket Adaptor.

In the multi-partition use-case it is required to access shared memory from different SoAd instances. To guarantee data consistency the data must be accessed atomically which is done by using the Bmc module [5] and its functions given in chapter 4.3. Partition-specific memory is used for the partition-specific functionality to assert that different partitions do not write to the same memory locations. This is done by generating partition-specific files. Partition-specific files exist per mapped OS Application and contain the short name of the OS application as naming convention. The files are listed in chapter 3.1.

5.3.4.19 Configurable TCP buffer segments

The Socket Adaptor stores the information about received TCP buffer segments in the TcpIp module to be able to continue TP-PDU receptions later if the user does not allow to copy all data in context of the reception itself.

The number of TCP segments which can be managed by Socket Adaptor per socket can be configured at:

`SoAd/SoAdGeneral/SoAdRxTcpBufferSegmentsMax`

Not all buffer segments which are announced via the `SoAd_RxIndication()` are managed as a single TCP segment. If the memory address indicates that the newly received TCP buffer segment is located directly behind the previously received TCP segment, the Socket Adaptor merges the TCP segments and treats them as one TCP segment.

The Socket Adaptor drops all TCP segments which cannot be managed.

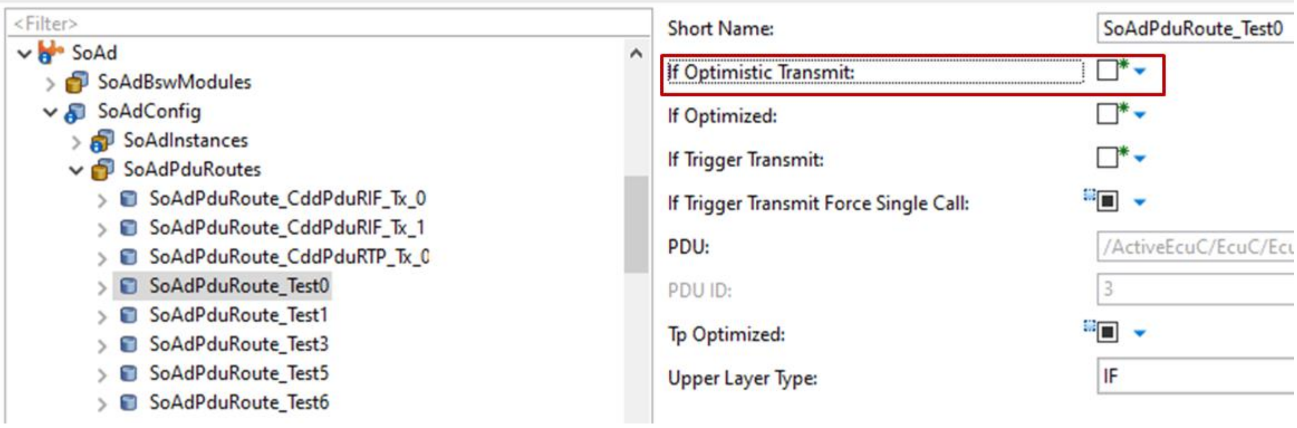
Caution
Configure `SoAdRxTcpBufferSegmentsMax` depending on the underlying TcpIp module to avoid data loss.

For MICROSAR Classic TcpIp, it is sufficient to configure the parameter to “2”.

5.3.4.20 Optimistic transmission fan-out

For PDU routes with IF upper layer, it is possible to configure several PDU route destinations (SWS_SoAd_00602). Moreover, according to AUTOSAR the Socket Adaptor shall return `E_NOT_OK` at `SoAd_IfTransmit()` in case the transmit request fails on any of the configured socket connections of the PDU route destinations (SWS_SoAd_00648).

In some applications it might make sense to enable the optimistic transmission fan-out feature. If the feature is enabled for a PDU route, `SoAd_IfTransmit()` will return `E_OK` when transmission succeeds on at least one of the related socket connections. The feature can be enabled or disabled (default value) for each configured IF PDU route as shown in Figure 5-34.



<Filter>

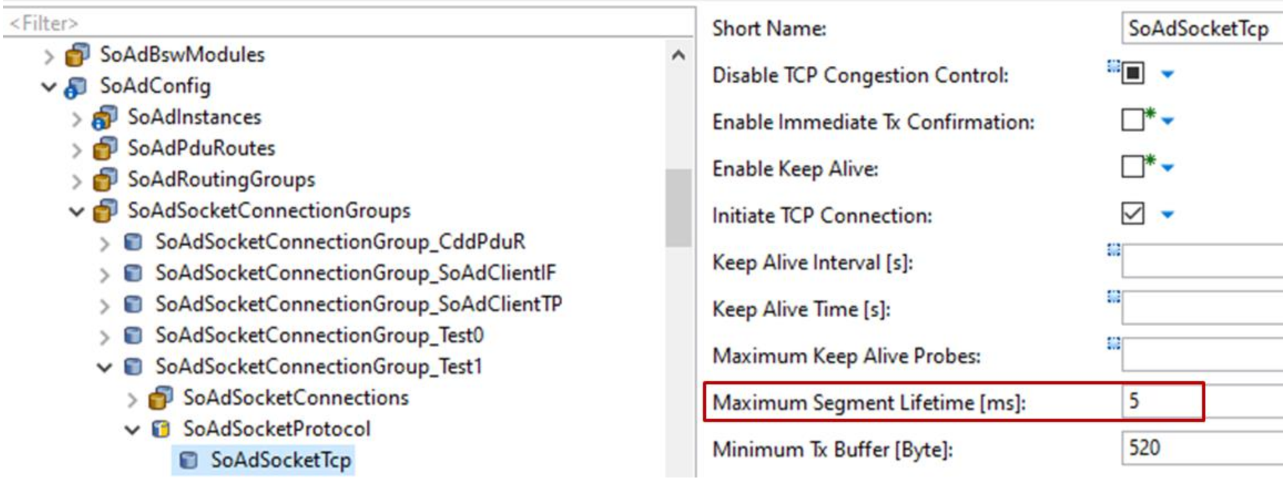
- SoAd
  - SoAdBswModules
  - SoAdConfig
    - SoAdInstances
    - SoAdPduRoutes
      - SoAdPduRoute_CddPduRIF_Tx_0
      - SoAdPduRoute_CddPduRIF_Tx_1
      - SoAdPduRoute_CddPduRTP_Tx_0
      - SoAdPduRoute_Test0
      - SoAdPduRoute_Test1
      - SoAdPduRoute_Test3
      - SoAdPduRoute_Test5
      - SoAdPduRoute_Test6

Short Name: SoAdPduRoute_Test0
If Optimistic Transmit: □
If Optimized: □
If Trigger Transmit: □
If Trigger Transmit Force Single Call: ■
PDU: /ActiveEcuC/EcuC/Ecu
PDU ID: 3
Tp Optimized: ■
Upper Layer Type: IF

Figure 5-34 IF optimistic transmission fan-out configuration

5.3.4.21 Socket specific MSL timeout

It is possible to enable a specific MSL timeout for TCP sockets by configuring `SoAdSocketTcpMsl` to the desired value as shown in Figure 5-35. This value is passed as integer in milliseconds to <TcpIp> component via `<TcpIp>_ChangeParameter` API and is used instead of the global `TcpIpTcpMsl` parameter for the corresponding socket.



<Filter>

- SoAdBswModules
- SoAdConfig
  - SoAdInstances
  - SoAdPduRoutes
  - SoAdRoutingGroups
  - SoAdSocketConnectionGroups
    - SoAdSocketConnectionGroup_CddPduR
    - SoAdSocketConnectionGroup_SoAdClientIF
    - SoAdSocketConnectionGroup_SoAdClientTP
    - SoAdSocketConnectionGroup_Test0
    - SoAdSocketConnectionGroup_Test1
      - SoAdSocketConnections
      - SoAdSocketProtocol
        - SoAdSocketTcp

Short Name: SoAdSocketTcp
Disable TCP Congestion Control: ■
Enable Immediate Tx Confirmation: □
Enable Keep Alive: □
Initiate TCP Connection: ☑
Keep Alive Interval [s]:
Keep Alive Time [s]:
Maximum Keep Alive Probes:
Maximum Segment Lifetime [ms]: 5
Minimum Tx Buffer [Byte]: 520

Figure 5-35 Socket specific MSL timeout configuration

Note
Socket specific MSL timeouts are not supported when Socket Adaptor is used together with vBsdAd module as lower layer.

5.3.4.22 Address resolution retry queue

On transmission, it is not ensured that the address resolution (ARP/NDP) has already succeeded. If the address resolution is still pending, no transmission is possible and transmission requests may be rejected.

The MICROSAR Classic TcpIp module implements a retry queue for UDP sockets that allows to store transmission requests if they would fail because of a pending address resolution. This queue can be enabled by Socket Adaptor on socket creation by enabling `SoAdSocketUdpAddressResolutionRetryEnabled`. The queue size can be configured by `SoAdSocketUdpAddressResolutionRetryQueueLimit`. Transmission requests are only accepted until this queue limit has been reached.

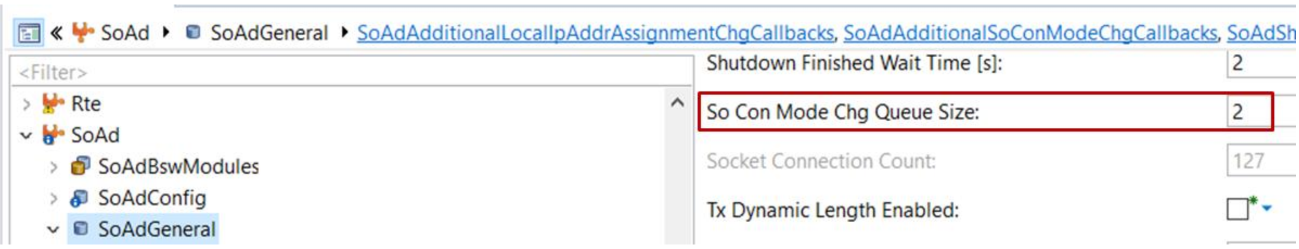
Caution
For memory and performance reasons, the TcpIp module stores the data to be transmitted in an Ethernet transmission buffer and does not implement an own buffer pool for the retry queue. Choose a proper value for `SoAdSocketUdpAddressResolutionRetryQueueLimit` to ensure that the Ethernet transmission buffers are not fully occupied by one socket and transmissions to other remote entities via other sockets are still possible.

5.3.4.23 SoConModeChg Queue

SoAd supports the configuration of a queue per socket connection to store the performed mode changes so that they can be reported in the correct order to the user. The mode changes are reported via callback as described in chapter 4.5.9. Also refer to chapter 5.3.4.4 for additional mode change notifications. Mode changes are performed in the following contexts:

- `SoAd_IfTransmit`
- `SoAd_TpTransmit`
- `SoAd_SetRemoteAddr`
- `SoAd_SetUniqueRemoteAddr`
- `SoAd_ReleaseRemoteAddr`
- `SoAd_ForceReleaseRemoteAddr`
- `SoAd_MainFunctionInstanceState`
- `SoAd_MainFunctionInstanceTx`
- `SoAd_RxIndication`
- `SoAd_TxConfirmation`
- `SoAd_TcpAccepted`
- `SoAd_TcpConnected`

In case the tasks of SoAd are configured in such a way that they may interrupt each other this may lead to interruptions before reporting the socket connection mode via `<Up>_SoConModeChg`. If the mode changes are expected to be reported in the right order the mode change queue mechanism must be enabled as shown in Figure 5-36 below. Usually, it is expected that a queue size of two is sufficient. If more interruptions are expected to occur at the same time the queue size may be increased. The queue elements are created per socket connection and whenever a mode change is performed the corresponding mode is written to the queue. The queue is handled immediately when changing the mode in case of no interruption. In case of interruption, the mode change reporting is delayed to the main function. For the main function handling and the limitation of elements handled per main function, please refer to the event queue handling described in chapter 5.3.4.12.1. When a mode change occurs while the queue has no free elements available a runtime error is reported if enabled (`Det_ReportRuntimeError` with the `ApiId` set to `SOAD_SID_SO_CON_MODE_CHG` and the `ErrorId` set to `SOAD_E_NO_MODE_CHG_QUEUE_ELEM`) and the last element in the queue is overwritten with the new mode to assert that the last reported mode corresponds to the mode in Socket Adaptor.


« SoAd ▸ SoAdGeneral ▸ SoAdAdditionalLocalIpAddrAssignmentChgCallbacks, SoAdAdditionalSoConModeChgCallbacks, SoAdSh
<Filter>
Rte
SoAd
SoAdBswModules
SoAdConfig
SoAdGeneral

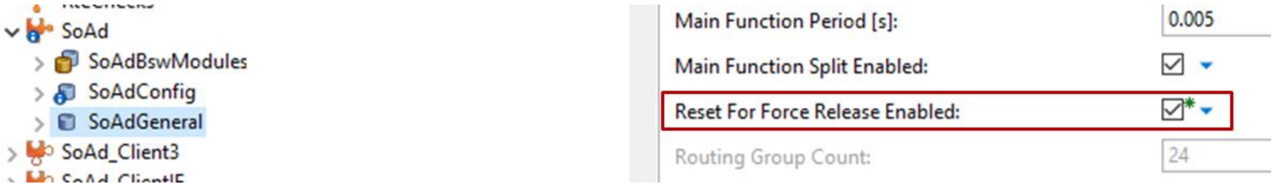
Shutdown Finished Wait Time [s]: 2
So Con Mode Chg Queue Size: 2
Socket Connection Count: 127
Tx Dynamic Length Enabled: □

Figure 5-36 SoConModeChg queue configuration

Please be aware that if the queueing mechanism is not used (queue size is set to zero), the mode in Socket Adaptor will always be correct independent of interruptions and only the reported order of mode changes may be incorrect.

5.3.4.24 Force Release Remote Address

Via `SoAd_ForceReleaseRemoteAddr` user can force the release of a remote address if the remote address is locked due to an open TCP socket connection. In this case the socket connection is requested to be reconnected. The closing behavior of the socket is configurable via BSWMD parameter `SoAdResetForForceReleaseEnabled` (see Figure 5-37). The default value is `TRUE`. This means that SoAd sends TCP RST when closing the socket (`abort` is `TRUE`). If the value is set to `FALSE` SoAd sends TCP FIN (`abort` is `FALSE`).



Main Function Period [s]: 0.005
Main Function Split Enabled: ☑
Reset For Force Release Enabled: ☑
Routing Group Count: 24

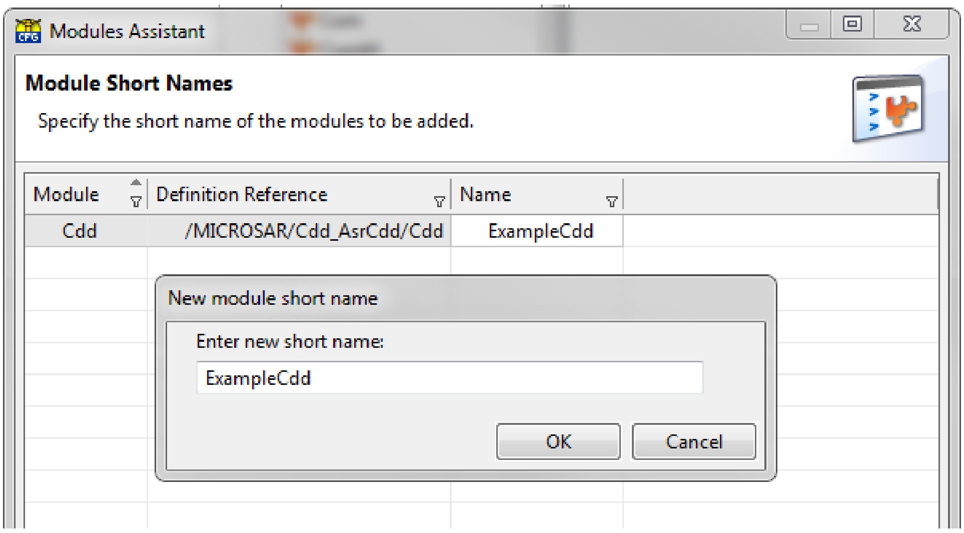
Figure 5-37 Force release remote address configuration

This is especially relevant for the Service Discovery use-case. In this case sending a TCP RST is recommended for reboot-detection as this leads to re-establishment of the connection.

5.3.5 Complex Device Driver (CDD)

Chapter 2.2.3.1 describes which CDDs are supported by Socket Adaptor.

The expected API prefix of CDD within Socket Adaptor depends on the name of CDD. The name can be chosen on creation of a CDD within “Module Assistant” of DaVinci Configurator Classic (Figure 5-38).



Module Short Names

Specify the short name of the modules to be added.

Module	Definition Reference	Name
Cdd	/MICROSAR/Cdd_AsrCdd/Cdd	ExampleCdd

New module short name

Enter new short name: ExampleCdd

Figure 5-38 CDD configuration

For the example in Figure 5-38 the prefix is `ExampleCdd`. The prototype for an IF RxIndication would be `ExampleCdd_[SoAd][If]RxIndication()`. `SoAd` and `If` infixes depends on configuration of “ExampleCdd” in “SoAdBswModules”.

6 Glossary and Abbreviations

6.1 Glossary

Term	Description
BSD Sockets	Berkeley sockets used by Linux for Ethernet communication
DaVinci Configurator Classic	Generation tool for MICROSAR Classic components

Table 6-1 Glossary

6.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
ARP	Address Resolution Protocol
AUTOSAR	Automotive Open System Architecture
BSD	Berkeley Software Distribution
BSW	Basis Software
CDD	Complex Device Driver (Complex Driver)
DEM	Diagnostic Event Manager
DET	Development Error Tracer
DoIP	Diagnostic over Internet Protocol
ECU	Electronic Control Unit
Eth	Ethernet Driver
EthTrcv	Ethernet Transceiver Driver
EthIf	Ethernet Interface
EthSM	Ethernet State Manager
FFI	Freedom from interference
IdsM	Intrusion Detection System Manager
IF	Interface API between BSW modules
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
MSL	Maximum Segment Lifetime
NDP	Neighbor Discovery Protocol
OS	Operating System
PduR	PDU Router
RTE	Runtime Environment
SchM	Schedule Manager
SoAd	Short name of Socket Adaptor
SRS	Software Requirement Specification
SwAddrMethods	Software address methods
SWS	Software Specification

TcpIp	Transport layer module containing TCP and UDP implementation
TCP	Transmission Control Protocol
TLS	Transport Layer Security
TP	Transport Protocol API between BSW modules
UDP	User Datagram Protocol
VLAN	Virtual Local Area Network

Table 6-2 Abbreviations

7 Contact

Visit our website for more information on

- News
- Products
- Demo software
- Support
- Training data
- Addresses

www.vector.com