VECTOR

MICROSAR Classic Diagnostic over IP

Technical Reference

Version 16.0.0

Authors	vismda, visjsb, vistia, vissem, visfaz, viseje, visgyv
Status	Released

Document Information

History

Author	Date	Version	Remarks
vismda	2014-05-26	1.0	Creation of document
vismda	2015-08-12	1.1	Update for component release
vismda	2016-01-11	2.0	Limited support of ASR4.2.2
visjsb, vismda	2016-02-25	2.1	Update for component release, UUDT Support according to ASR 4.2.2
vismda	2016-05-19	2.2	Improved Vehicle Announcement Handling
vismda	2016-09-05	3.0	Support of OEM specific payload types, Target Address Masking and Verification, Optimized TP transmission
vismda	2017-01-30	3.1	Support multiple testers on different VLANs
vismda	2017-05-08	4.0	Update component history
vismda	2017-05-22	4.1	Update component history
vismda	2017-07-04	4.2	PDU reception verification
visjsb	2017-07-20	4.3	API Pattern, Cleanup
visjsb	2017-08-21	4.4	Update component history
visjsb	2017-09-25	5.0	Update critical sections
visjsb	2018-01-31	5.1	Update component history
visjsb	2018-02-23	5.2	Update function declaration
vismda	2018-04-13	6.0	Support Activation Line for IPv6
vismda	2018-05-23	6.1	Support Activation Line for automatically assigned IP addresses
visjsb	2018-10-11	7.0	Update APIs, DHCP vendor option
visjsb	2019-06-24	7.1	Update component history
vistia	2019-08-06	8.0	Added support for API to retrieve and reset measurement data
vismda, visjsb	2019-10-24	9.0	Update figure for interfaces to adjacent modules, update reentrance
vissem	2021-03-09	10.0	Added support for API to overwrite the logical target address
visfaz, vissem	2021-03-26	11.0	Specified buffer lengths for callout functions Added DoIP Interfaces
vissem, viseje	2021-04-20	11.1	Activation line per DoIP Interface, Entity status per local interface and address
visfaz	2021-05-08	11.2	Optional vehicle announcement, Trigger vehicle announcement

vismda	2021-06-25	11.3	Updated AUTOSAR specification analysis to R19-11
viseje, visjsb	2021-07-09	11.4	Improved description of shutdown feature, Adapt description of pdu size routing feature
visjsb	2021-10-18	12.0	Soft activation line state change
viseje	2022-01-25	12.1	Provide received data in StartOfReception, Usage of DoIP_MemMap.h
viseje, visjsb	2022-03-08	13.0	Product name updated to MICROSAR Classic, Optional n/k+1 Socket
viseje, vismda	2022-06-09	13.1	Enable DoIP Routing Activation on TCP to detect unused TLS Connections
viseje	2023-02-14	14.0	Extend OEM specific payload type solution by transmit API
visgyv	2023-04-18	15.0	Provide channel ready for transmission information for Dcm
viseje	2023-05-11	15.1	Support Postbuild Selectable and Loadable
viseje	2023-07-03	15.2	Make DoIP Protocol Version configurable, Improved documentation for "Channel Ready Notification"
visjsb	2023-09-06	15.3	Extend DoIP Alive Check to detect unused TLS Connections earlier
visjsb	2024-01-30	16.0	Deactivation of TLS for DoIP by Application Code

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	Specification of Diagnostic over IP	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]	ISO	ISO 13400-2	IS 2012
[6]	ISO	ISO 13400-2	IS 2019
[7]	Vector	User Manual Post-Build Loadable	See delivery
[8]	Vector	User Manual Identity Manager	See delivery
[9]	Vector	TechnicalReference MICROSAR Classic DCM	See delivery

Scope of the Document

This Technical Reference describes the general use of the DoIP basis software module. 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 DoIP as specified in [1].

Supported Configuration Variants:	pre-compile, post-build	
Vendor ID:	DoIP_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	DoIP_MODULE_ID	173 decimal (according to ref. [4])

The DoIP module is a protocol to transport diagnostic data over Ethernet from tester to ECU and vice versa. It is also described in ISO Standard 13400.

Following key features are offered by DoIP:

- Vehicle Identification to detect DoIP supporting entities
- Node information to get information and states from entities
- Routing Activation to register tester on an entity
- Request and Acknowledge behavior for diagnostic data for advanced protocol handling
- Timeout and alive mechanism to maintain socket and tester connections

1.1 Architecture Overview

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

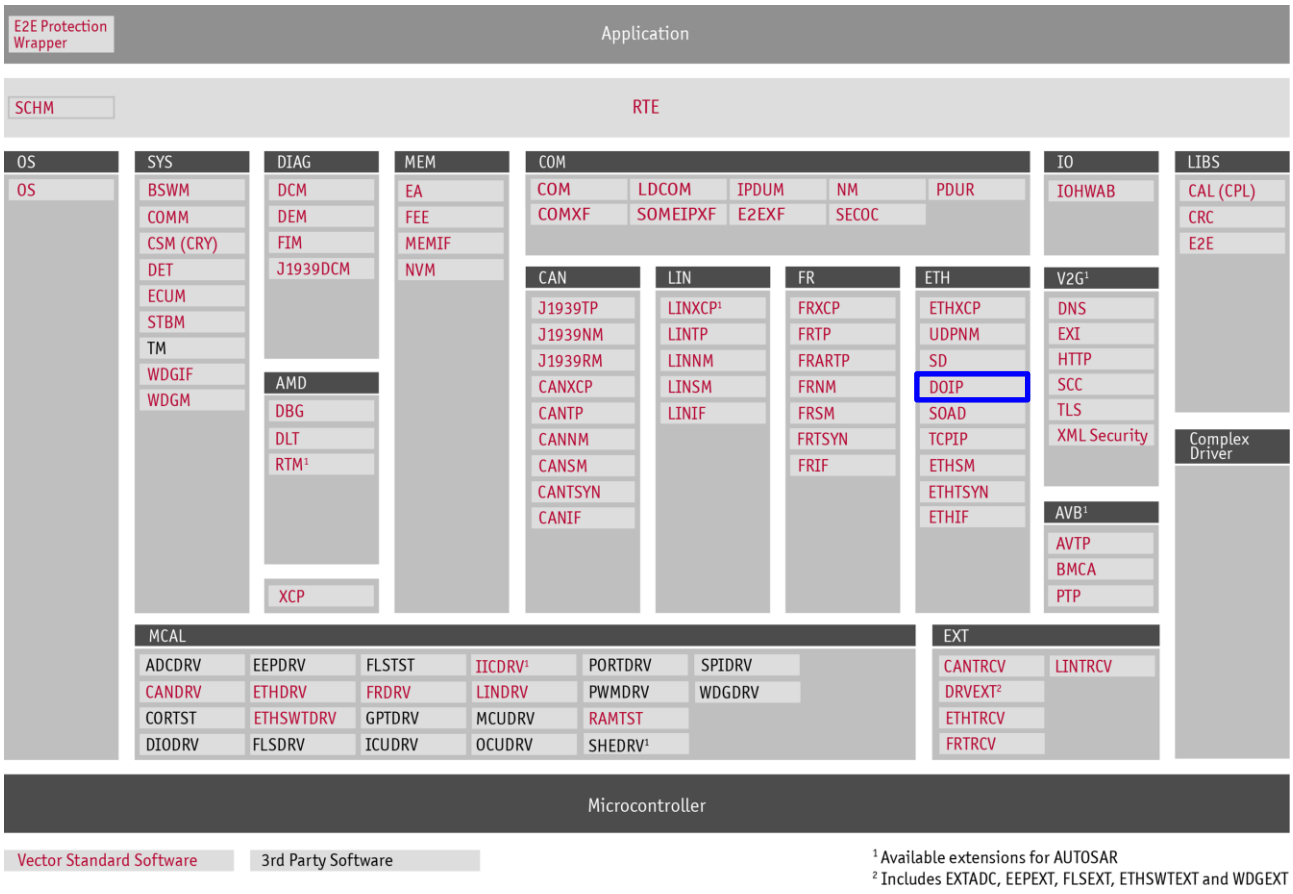

Figure 1-1 AUTOSAR 4.2 Architecture Overview

The next figure shows the interfaces to adjacent modules of the DoIP. These interfaces are described in chapter 4.

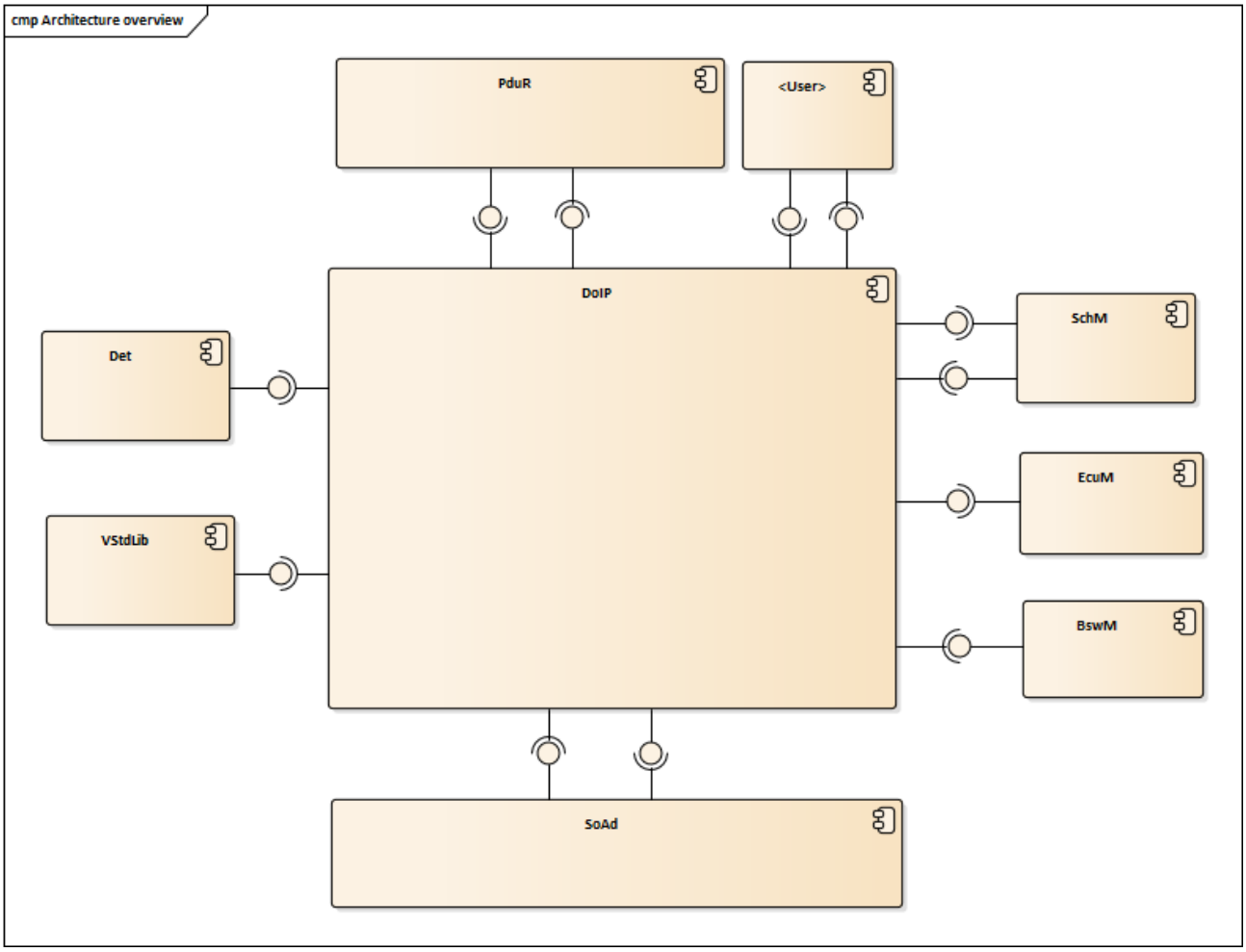

Figure 1-2 Interfaces to adjacent modules of the DoIP

2 Functional Description

2.1 Features

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

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

> Table 2-1 Supported AUTOSAR standard conform features

> Table 2-2 Not supported AUTOSAR standard conform features

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

> Table 2-3 Features provided beyond the AUTOSAR standard

The following features specified in [1] are supported:

Supported AUTOSAR Standard Conform Features
DoIP Message layout according ISO 13400-2
UDP/TCP communication and socket handling
Tester acceptance depending on logical address
DoIP Channels (routing depending on logical target address)
Diagnostic Message (i.e. user data) within Diagnostic Message Acknowledge Message
Interface specific DoIP Activation Line
Routing Activation handling for OEM specific part
UUDT over IF-API
Secure connections
Support of post-build loadable
Support of post-build selectable

Table 2-1 Supported AUTOSAR standard conform features

2.1.1 Deviations

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

Category	Description
Functional	Instead of negative acknowledge code 0x08, the code 0x05 is sent if PduR_DoIPTpStartOfReception fails.
API	DoIP Service Component is not supported.
API	The service IDs are as specified in R4.2.2.
API	<User>_DoIPGetFurtherByteCallback is not called.
API	<User>_DoIPRoutingActivationConfirmation is called as specified in R4.2.2 and does not expect constant pointer.

Category	Description
API	<User>_DoIPRoutingActivationAuthentication does not expect constant pointer.
API	DoIP_SoAdTpStartOfReception does not implement constant pointer.
API	DoIP_SoAdTpCopyRxData does not implement constant pointer.
API	DoIP_SoAdTpCopyTxData does not implement constant pointer.
API	DoIP_SoAdIfTxConfirmation has no parameter “Result” (refer to R4.2.2).
Config	DoIPHostNameSizeMax is a string parameter (refer to R4.2.2).
Config	DoIPFurtherActionByteCallback is not supported.

Table 2-2 Not supported AUTOSAR standard conform features

2.1.2 Additions/ Extensions

The following features are provided beyond the AUTOSAR standard:

Features Provided Beyond the AUTOSAR Standard
PDU Size Routing
Maximum message size for each DoIP Channel
Default Tester
Multiple local IP addresses per interface
Shutdown mechanism
OEM specific payload types
Target Address Masking and Verification
Optimized TP transmission
PDU reception verification
Disable control of IP address assignment
DHCP option field to be used for target IP addresses of DoIP
Get and reset measurement data
Overwrite logical target address during runtime
Soft activation line state change
Graceful close
Configurable amount of user data to be passed on StartOfReception
Optional n/k+1 Socket
Manual closing of a connection
Provide channel ready to Dcm
Configurable DoIP protocol version
Alive Checks before General Inactivity Time is reached

Table 2-3 Features provided beyond the AUTOSAR standard

2.1.3 Known Issues (Low Priority)

There are no known issues.

2.1.4 Hints

2.1.4.1 API deviation

The API to PduR and SoAd is implemented partly according to AUTOSAR 4.1.3 and 4.2.2. Please refer to chapter 4 for details.

2.1.4.2 Routing Activation Handler

There is exactly one routing activation handler implemented. If activation handler is occupied on reception of a routing activation request, all newly received routing activation requests will remain in TcpIp buffer. After handler has been released, the next routing activation request is handled.

Caution
If authentication or confirmation is pending (e.g. `DOIP_E_PENDING` is returned by call to callback <User>_DoIPRoutingActivationAuthentication) routing activation handler is not released until different value is returned or socket is closed.

2.2 Initialization

DoIP is initialized via a call of `DoIP_InitMemory()` followed by the call of `DoIP_Init()`. A call to `DoIP_InitMemory()` reverts the initialization and a call to `DoIP_Init()` is required to initialize DoIP again.

2.2.1 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 DoIP module has to be initialized using the DoIP_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 DoIP_Lcfg.h and DoIP_Lcfg.c. For Post-build selectable the pointer to the required variant data needs to be passed.

2.2.2 Configuration Variant 4 (Post-build loadable)

In this configuration Variant, the DoIP module has to be initialized using the DoIP_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 DoIP_PBcfg.h and DoIP_PBcfg.c. Please refer to chapter 5.1 to get information about supported configuration variants.

2.3 States

The DoIP has no specific state handling after calling the initialization functions described in chapter 2.2.
Please note that the module provides a shutdown mechanism (chapter 5.3.2.6).

2.4 Main Functions

Within the main functions DoIP handles:

> Socket connection handling

- Vehicle Announcements
- Alive Check
- Initial and General Inactivity timer
- TCP Transmission Queue for parallel transmission requests on one socket
- UDP Transmission Queue for retries
- Routing Activation Authentication and Confirmation

2.5 Error Handling

2.5.1 Development Error Reporting

By default, development errors are reported to the DET using the service `Det_ReportError()` as specified in [2], if the development error reporting is enabled (i.e. pre-compile parameter `DOIP_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 DoIP ID according to AUTOSAR is 173.

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
0x00	DoIP_GetVersionInfo()
0x01	DoIP_Init()
0x02	DoIP_MainFunction()
0x03	DoIP_TpTransmit()
0x04	DoIP_TpCancelTransmit()
0x05	DoIP_TpCancelReceive()
0x0B	DoIP_SoConModeChg()
0x0C	DoIP_LocalIpAddrAssignmentChg()
0x0D	DoIP_TriggerVehicleAnnouncement()
0x0E	DoIP_ActivationLineSwitch()
0x40	DoIP_SoAdIfTxConfirmation()
0x42	DoIP_SoAdIfRxIndication()
0x43	DoIP_SoAdTpCopyTxData()
0x44	DoIP_SoAdTpCopyRxData()
0x45	DoIP_SoAdTpRxIndication()
0x46	DoIP_SoAdTpStartOfReception()

Service ID	Service
0x48	DoIP_SoAdTpTxConfirmation()
0x49	DoIP_IfTransmit()
0x4A	DoIP_IfCancelTransmit()
0xEC	DoIP_EnablePduSizeRouting()
0xED	DoIP_DisablePduSizeRouting()
0xEF	DoIP_TxTcp_Transmit()
0XF0	DoIP_ShutdownFinished()
0xF1	DoIP_DhcpEvent()
0xF2	DoIP_GetAndResetMeasurementData()
0xF3	DoIP_OverwriteLogicalTargetAddress()
0xF4	DoIP_TransmitOemSpecificPayloadType()
0xF5	DoIP_CloseConnection()

Table 2-4 Service IDs

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

Error Code	Description
0x00	DOIP_E_NO_ERROR
0x01	DOIP_E_UNINIT
0x02	DOIP_E_PARAM_POINTER
0x03	DOIP_E_INVALID_PDU_SDU_ID
0x04	DOIP_E_INVALID_PARAMETER
0x05	DOIP_E_INIT_FAILED
0xEC	DOIP_E_ACTIVATION_LINE
0xED	DOIP_E_SOAD_CALL_FAILED
0xEE	DOIP_E_UNEXPECTED_ASSIGNMENT
0xEF	DOIP_E_NOBUFS
0xF0	DOIP_E_PDU_SIZE_ROUTING
0xF1	DOIP_E_SHUTDOWN_FINISHED
0xF2	DOIP_E_DHCP_EVENT

Table 2-5 Errors reported to DET

2.5.2 Production Code Error Reporting

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

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

DoIP does not report any production errors.

3 Integration

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

3.1 Embedded Implementation

The delivery of the DoIP module contains these source code files:

File Name	Description	Integration Task
DoIP.c	Source file.	-
DoIP.h	Header file.	-
DoIP_Cbk.h	Header file for callback functions.	-
DoIP_Priv.h	Header file for component intern declarations and definitions.	-
DoIP_Types.h	Header file for types.	-
_Appl_DoIP.c	Template source file for callout functions.	Adapt the template file. See hints within that file.
_Appl_DoIP.h	Template header file for callout functions.	Adapt the template file. See hints within that file.
DoIP_Lcfg.c	Generated source file.	-
DoIP_Lcfg.h	Generated header file.	-
DoIP_PBcfg.c	Generated source file (Post-build configuration)	-
DoIP_PBcfg.h	Generated header file (Post-build configuration)	-
DoIP_Cfg.h	Header file for configuration parameters and defines (e.g. feature switches).	-
DoIP_MemMap.h	Generated header file containing the memory sections of DoIP	

Table 3-1 Implementation files

3.2 Critical Sections

All services and callbacks for transmission, reception and state changes of DoIP 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 DoIP context are interrupt services originated from physical bus events (Ethernet, CAN, LIN, FlexRay etc.).

Relevant tasks in the DoIP context are all tasks which call DoIP API functions. Usually these tasks are limited to tasks on which other BSW modules (SoAd, PduR, DCM, etc.) are mapped to.

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

> DOIP_EXCLUSIVE_AREA_0 is used whenever memory accesses must be protected from accesses of interrupting calls to services and callbacks of DoIP. 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 long.

For an implementation of the critical section it could be sufficient to

> Disable all bus relevant interrupts of all buses related to calls to DoIP API functions.

> Disable all Ethernet bus relevant interrupts if all modules calling DoIP API functions are mapped to one task (e.g. SchM task) or a non-preemptive OS is 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.

3.3 Main Functions

DoIP 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 DoIP related modules are the SoAd and the TcpIp module.

Caution
Consider main function expectations when using preemptive tasks.

4 API Description

For an interface overview please see Figure 1-2.

4.1 Type Definitions

The types defined by the DoIP are described in this chapter.

Type Name	C-Type	Description	Value Range
DoIP_ConfigType	uint8	Module configuration	
DoIP_PowerStateType	uint8	Power mode	0x00u DOIP_POWER_MODE_NOT_READY
			0x01u DOIP_POWER_MODE_READY
			0x02u DOIP_POWER_MODE_NOT_SUPPORTED
DoIP_OemPayloadTypeFlagType	uint8	Reception flags for manufacturer-specific payload types	Bit0: 0=UDP 1=TCP Bit1: 0=no routing activation 1=routing activation
DoIP_MeasurementIdxType	uint8	Index to select specific measurement data	0x01u DOIP_MEAS_DROP_TCP 0x02u DOIP_MEAS_DROP_UDP 0xFFu DOIP_MEAS_ALL
DoIP_ChannelIdType	uint16	Unique channel ID	
DoIP_ConnectionIdType	uint16	Unique connection ID	

Table 4-1 Type definitions

4.2 Services provided by DoIP

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

4.2.1 DoIP_InitMemory

Prototype	
`void DoIP_InitMemory (void)`	
Parameter	
void	none

Return code	
void	none
Functional Description	
Power-up memory initialization. Initializes component variables in *_INIT_* sections at power up.	
Particularities and Limitations	
Use this function in case these variables are not initialized by the startup code. Module is uninitialized.	
Call context	
> TASK > This function is Synchronous > This function is Non-Reentrant	

Table 4-2 DoIP_InitMemory

4.2.2 DoIP_Init

Prototype	
`void DoIP_Init (const DoIP_ConfigType *DoIPConfigPtr)`	
Parameter	
DoIPConfigPtr [in]	Pointer to the configuration data of the DoIP module.
Return code	
void	none
Functional Description	
Initializes component. Initializes all component variables and sets the component state to initialized. This service initializes all global variables of the DoIP module. After return of this service the DoIP module is operational.	
Particularities and Limitations	
> Interrupts are disabled.Module is uninitialized.DoIP_InitMemory has been called unless DoIP_State is initialized by start-up code.	
Call context	
> TASK > This function is Synchronous > This function is Non-Reentrant	

Table 4-3 DoIP_Init

4.2.3 DoIP_GetVersionInfo

Prototype	
`void DoIP_GetVersionInfo (Std_VersionInfoType *versioninfo)`	
Parameter	
versioninfo [out]	Pointer to where to store the version information of this module. 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): DOIP_VERSION_INFO_API	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Reentrant	

Table 4-4 DoIP_GetVersionInfo

4.2.4 DoIP_TpTransmit

Prototype	
`Std_ReturnType` **`DoIP_TpTransmit`** `(PduIdType DoIPPduRTxId, const PduInfoType *DoIPPduRTxInfoPtr)`	
Parameter	
DoIPPduRTxId [in]	DoIP unique identifier of the Pdu to be transmitted by the PduR.
DoIPPduRTxInfoPtr [in]	Tx Pdu information structure which contains the length of the DoIPTxMessage.
Return code	
Std_ReturnType	E_OK The request has been accepted.
Std_ReturnType	E_NOT_OK The request has not been accepted, e.g. parameter check has failed or no resources are available for transmission.
Functional Description	
Requests transmission of a specific TP Pdu. This service is called to request the transfer data from the PduRouter to the SoAd. It is used to indicate the transmission which will be performed by SoAd or in the DoIP_Mainfunction. Within the provided DoIPPduRTxInfoPtr only SduLength is valid (no data)! If this function returns E_OK then the SoAd module will raise a subsequent call to PduR_DoIPCopyTxData via DoIP_SoAdTpCopyRxData in order to get the data to send.	
Particularities and Limitations	
-	
Call context	
> TASK > This function is Non-Reentrant for the same ID but Reentrant for different IDs.	

Table 4-5 DoIP_TpTransmit

4.2.5 DoIP_TpCancelTransmit

Prototype	
`Std_ReturnType` **`DoIP_TpCancelTransmit`** `(PduIdType DoIPPduRTxId)`	
Parameter	
DoIPPduRTxId [in]	DoIP unique identifier of the Pdu to be canceled by the PduR.
Return code	
Std_ReturnType	E_OK Transmit cancellation request of the specified DoIPPduRTxId is accepted.
Std_ReturnType	E_NOT_OK The transmit cancellation request of the DoIPPduRTxId has been rejected.
Functional Description	
Requests transmission cancellation of a specific TP Pdu. This service primitive is used to cancel the transfer of pending DoIPPduRTxIds. The connection is identified by DoIPPduRTxId. If the function returns the cancellation is requested but not yet performed.	
Particularities and Limitations	
-	
Call context	
> TASK > This function is Reentrant	

Table 4-6 DoIP_TpCancelTransmit

4.2.6 DoIP_TpCancelReceive

Prototype	
`Std_ReturnType` **`DoIP_TpCancelReceive`** `(PduIdType DoIPPduRRxId)`	
Parameter	
DoIPPduRRxId [in]	DoIP unique identifier of the Pdu for which reception shall be canceled by the PduR.
Return code	
Std_ReturnType	E_OK Reception was canceled successfully.
Std_ReturnType	E_NOT_OK Reception was not canceled.
Functional Description	
Requests reception cancellation of a specific TP Pdu. By calling this API with the corresponding DoIPPduRRxId the currently ongoing data reception is terminated immediately. If the function returns the cancellation is requested but not yet performed.	
Particularities and Limitations	
-	
Call context	
> TASK > This function is Reentrant	

Table 4-7 DoIP_TpCancelReceive

4.2.7 DoIP_IfTransmit

Prototype	
`Std_ReturnType DoIP_IfTransmit (PduIdType id, const PduInfoType *info)`	
Parameter	
id [in]	Identification of the IF Pdu.
info [in]	Length and pointer to the buffer of the IF Pdu.
Return code	
Std_ReturnType	E_OK Request is accepted by the destination module.
Std_ReturnType	E_NOT_OK Request is not accepted by the destination module.
Functional Description	
Requests transmission of a specific IF Pdu. -	
Particularities and Limitations	
-	
Call context	
> TASK > This function is Synchronous > This function is Non-Reentrant for the same ID but Reentrant for different IDs.	

Table 4-8 DoIP_IfTransmit

4.2.8 DoIP_IfCancelTransmit

Prototype	
`Std_ReturnType DoIP_IfCancelTransmit (PduIdType id)`	
Parameter	
id [in]	Identification of the IF Pdu to be cancelled.
Return code	
Std_ReturnType	E_OK Cancellation was executed successfully by the destination module.
Std_ReturnType	E_NOT_OK Cancellation was rejected by the destination module.
Functional Description	
Requests transmission cancellation of a specific IF Pdu. -	
Particularities and Limitations	
-	
Call context	

> TASK > This function is Reentrant

Table 4-9 DoIP_IfCancelTransmit

4.2.9 DoIP_TransmitOemSpecificPayloadType

Prototype	
`Std_ReturnType` **`DoIP_TransmitOemSpecificPayloadType`** `(DoIP_ConnectionIdType ConnectionId, uint16 PayloadType, const PduInfoType *PayloadDataPtr)`	
Parameter	
ConnectionId [in]	DoIP unique identifier of the connection.
PayloadType [in]	The payload type that shall be used for the message.
PayloadDataPtr [in]	PDU information structure which contains the length and the payload data of the message.
Return code	
Std_ReturnType	E_OK Transmit request was accepted.
Std_ReturnType	E_NOT_OK Transmit request was not accepted.
Functional Description	
Requests transmission of a message with OEM specific payload type. -	
Particularities and Limitations	
-	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Non-Reentrant for the same ID but Reentrant for different IDs.	

Table 4-10 DoIP_TransmitOemSpecificPayloadType

4.2.10 DoIP_EnablePduSizeRouting

Prototype	
`void` **`DoIP_EnablePduSizeRouting`** `(void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
Activates the DoIP packet size dependent routing. -	
Particularities and Limitations	
-	

Configuration Variant(s): DOIP_SUPPORT_PDU_SIZE_ROUTING
Call context
> TASK > This function is Synchronous > This function is Reentrant

Table 4-11 DoIP_EnablePduSizeRouting

4.2.11 DoIP_DisablePduSizeRouting

Prototype	
`void DoIP_DisablePduSizeRouting (void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
Deactivates the DoIP packet size dependent routing. -	
Particularities and Limitations	
- Configuration Variant(s): DOIP_SUPPORT_PDU_SIZE_ROUTING	
Call context	
> TASK > This function is Synchronous > This function is Reentrant	

Table 4-12 DoIP_DisablePduSizeRouting

4.2.12 DoIP_Shutdown

Prototype	
`Std_ReturnType DoIP_Shutdown (void)`	
Parameter	
void	none
Return code	
Std_ReturnType	E_OK Shutdown request was accepted.
	E_NOT_OK Shutdown request was not accepted.
	SOAD_E_INPROGRESS Shutdown is in progress.
Functional Description	
Shutdown of SoAd. All sockets will be closed and modules change to special shutdown state.	

Particularities and Limitations
-
Configuration Variant(s): DOIP_SUPPORT_SHUTDOWN
Call context
> TASK
> This function is Non-Reentrant

Table 4-13 DoIP_Shutdown

4.2.13 DoIP_GetAndResetMeasurementData

Prototype	
`Std_ReturnType DoIP_GetAndResetMeasurementData (DoIP_MeasurementIdxType MeasurementIdx, boolean MeasurementResetNeeded, uint32* MeasurementDataPtr)`	
Parameter	
MeasurementIdx [in]	Data index of measurement data.
MeasurementResetNeeded [in]	Flag to trigger a reset of the measurement data.
MeasurementDataPtr [out]	Reference to data buffer, where to copy measurement data.
Return code	
Std_ReturnType	E_OK Measurement data retrieved and/or reset successfully.
	E_NOT_OK Measurement data retrieval and/or reset failed.
Functional Description	
Returns the measurement data for the specified measurement Index. Additionally, it resets the measurement data after the read operation, if specified to do so. The status of the operation is specified in the return value and the data is copied to the data pointer passed into the function as a parameter.	
Particularities and Limitations	
-	
Configuration Variant(s): DOIP_GET_RESET_MEASURE_DATA	
Call context	
> TASK | ISR2	
> This function is Synchronous	
> This function is Non-Reentrant	

Table 4-14 DoIP_GetAndResetMeasurementData

4.2.14 DoIP_OverwriteLogicalTargetAddress

Prototype	
`Std_ReturnType DoIP_OverwriteLogicalTargetAddress (DoIP_ChannelIdType ChannelId, uint16 LogicalTargetAddress)`	
Parameter	
ChannelId [in]	DoIP unique identifier of the channel.
LogicalTargetAddress [in]	The new logical target address

Return code	
Std_ReturnType	E_OK Logical target address successfully overwritten.
	E_NOT_OK Failed to overwrite the logical target address.
Functional Description	
Overwrites the logical target address of the passed channel. -	
Particularities and Limitations	
- Configuration Variant(s): DOIP_OVERWRITE_LOGICAL_TARGET_ADDR_API	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Non-Reentrant	

Table 4-15 DoIP_OverwriteLogicalTargetAddress

4.2.15 DoIP_ActivationLineSwitch

Prototype	
`void DoIP_ActivationLineSwitch (uint8 InterfaceId, boolean active)`	
Parameter	
InterfaceId[in]	DoIP unique identifier of the interface.
active[in]	Flag to indicate the state of the activation line.
Return code	
void	none
Functional Description	
Notifies DoIP on a switch of the DoIPActivationLine. This function is used to notify the DoIP on a switch of the DoIPActivationLine of the DoIP Interface with the given InterfaceId.	
Particularities and Limitations	
-	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Non-Reentrant > The application must ensure that there are not more than 4294967295 ((2^32) - 1) pending API calls.	

Table 4-16 DoIP_ActivationLineSwitch

4.2.16 DoIP_CloseConnection

Prototype	
`Std_ReturnType DoIP_CloseConnection (DoIP_ConnectionIdType ConnectionId, boolean Abort)`	
Parameter	
ConnectionId [in]	DoIP unique identifier of the connection.
Abort [in]	Flag to close connection immediately. [range: TRUE close immediately, FALSE close when no transmission is pending]
Return code	
Std_ReturnType	E_OK Request was accepted.
Std_ReturnType	E_NOT_OK Request was not accepted.
Functional Description	
Closes the requested connection. -	
Particularities and Limitations	
-	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Non-Reentrant for the same ID but Reentrant for different IDs.	

Table 4-17 DoIP_CloseConnection

4.2.17 DoIP_TriggerVehicleAnnouncement

Prototype	
`void DoIP_TriggerVehicleAnnouncement (uint8 InterfaceId)`	
Parameter	
InterfaceId[in]	DoIP unique identifier of the interface.
Return code	
void	none
Functional Description	
Notifies DoIP to start vehicle announcement. This function is used to notify the DoIP module to start vehicle announcement for DoIP interfaces with given InterfaceId.	
Particularities and Limitations	
-	
Call context	
> TASK|ISR2 > This function is Non-Reentrant	

Table 4-18 DoIP_TriggerVehicleAnnouncement

4.2.18 DoIP_MainFunction

Prototype	
`void DoIP_MainFunction (void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
Issue vehicle announcement, alive check and inactivity timeout handling.	
Particularities and Limitations	
Call context	
> TASK > This function is Synchronous > This function is Non-Reentrant	

Table 4-19 DoIP_MainFunction

4.3 Services used by DoIP

In the following table services provided by other components, which are used by the DoIP are listed. For details about prototype and functionality refer to the documentation of the providing component.

Component	API
Det	Det_ReportError
VStdLib	VStdLib_MemCpy
PduR	PduR_DoIPCopyRxData
PduR	PduR_DoIPCopyTxData
PduR	PduR_DoIPIfTxConfirmation
PduR	PduR_DoIPTpCopyRxData
PduR	PduR_DoIPTpCopyTxData
PduR	PduR_DoIPTpRxIndication
PduR	PduR_DoIPTpStartOfReception
PduR	PduR_DoIPTpTxConfirmation
PduR	PduR_SoAdTpCopyRxData
SoAd	SoAd_CloseSoCon
SoAd	SoAd_GetLocalAddr
SoAd	SoAd_GetPhysAddr
SoAd	SoAd_GetRemoteAddr
SoAd	SoAd_GetSoConId

Component	API
SoAd	SoAd_IfTransmit
SoAd	SoAd_OpenSoCon
SoAd	SoAd_ReleaseIpAddrAssignment
SoAd	SoAd_RequestIpAddrAssignment
SoAd	SoAd_Shutdown
SoAd	SoAd_TpCancelReceive
SoAd	SoAd_TpCancelTransmit
SoAd	SoAd_TpTransmit
SoAd	SoAd_WriteDhcpHostNameOption
SoAd	SoAd_WriteDhcpOption
SoAd	SoAd_ReadDhcpOption
BswM	BswM_DoIP_SetChannelReady
EcuM	EcuM_BswErrorHook

Table 4-20 Services used by the DoIP

4.4 Callback Functions

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

4.4.1 DoIP_SoAdTpCopyTxData

Prototype	
`BufReq_ReturnType` **`DoIP_SoAdTpCopyTxData`** `(PduIdType id, PduInfoType *info, RetryInfoType *retry, PduLengthType *availableDataPtr)`	
Parameter	
id [in]	Identification of the transmitted TP Pdu.
info [in]	Provides the destination buffer (SduDataPtr) and the number of bytes to be copied (SduLength). An SduLength of 0 can be used to query the current amount of available data in the upper layer module. In this case, the SduDataPtr may be a NULL_PTR.
retry [in]	Retry is not supported by SoAd (NULL_PTR)
availableDataPtr [out]	Indicates the remaining number of bytes that are available in the upper layer module's Tx buffer.
Return code	
BufReq_ReturnType	BUFREQ_OK Data has been copied to the transmit buffer completely as requested.
BufReq_ReturnType	BUFREQ_E_NOT_OK Data has not been copied. Request failed.
Functional Description	
Queries transmit data of a Pdu. This function is called to acquire the transmit data of an I-PDU segment (N-PDU). Each call to this function provides the next part of the TP Pdu data. The size of the remaining data is written to the position indicated	

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

Table 4-21 DoIP_SoAdTpCopyTxData

4.4.2 DoIP_SoAdTpTxConfirmation

Prototype	
`void DoIP_SoAdTpTxConfirmation (PduIdType id, Std_ReturnType result)`	
Parameter	
id [in]	Identification of the transmitted TP Pdu.
result [in]	Result of the transmission of the TP Pdu. Range: NTFRSLT_OK, NTFRSLT_E_NOT_OK.
Return code	
void	none
Functional Description	
Transmission confirmation callback for TP. This function is called after the TP Pdu has been transmitted on its network, the result indicates whether the transmission was successful or not.	
Particularities and Limitations	
-	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Non-Reentrant for the same ID but Reentrant for different IDs.	

Table 4-22 DoIP_SoAdTpTxConfirmation

4.4.3 DoIP_SoAdTpCopyRxData

Prototype	
`BufReq_ReturnType DoIP_SoAdTpCopyRxData (PduIdType id, PduInfoType *info, PduLengthType *bufferSizePtr)`	
Parameter	
id [in]	Identification of the received TP Pdu.
info [in]	Provides the source buffer (SduDataPtr) and the number of bytes to be copied (SduLength). An SduLength of 0 can be used to query the current amount of available buffer in the upper layer module. In this case, the SduDataPtr may be a NULL_PTR.

bufferSizePtr [out]	Available receive buffer after data has been copied.
Return code	
BufReq_ReturnType	BUFREQ_OK Data copied successfully.
BufReq_ReturnType	BUFREQ_E_NOT_OK Data was not copied because an error occurred.
Functional Description	
Called when SoAd has data to copy. This function is called to provide the received data of an TP Pdu segment (N-PDU) to the upper layer. Each call to this function provides the next part of the TP Pdu data. The size of the remaining data is written to the position indicated by bufferSizePtr.	
Particularities and Limitations	
-	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Non-Reentrant for the same ID but Reentrant for different IDs.	

Table 4-23 DoIP_SoAdTpCopyRxData

4.4.4 DoIP_SoAdTpStartOfReception

Prototype	
`BufReq_ReturnType` **`DoIP_SoAdTpStartOfReception`** `(PduIdType id, PduInfoType *info, PduLengthType TpSduLength, PduLengthType *bufferSizePtr)`	
Parameter	
id [in]	Identification of the TP Pdu.
info [in]	Pointer to data to support first or single frames (not used and ignored)
TpSduLength [in]	Total length of the N-SDU to be received.
bufferSizePtr [out]	Available receive buffer in the receiving module.
Return code	
BufReq_ReturnType	BUFREQ_OK Connection has been accepted. bufferSizePtr indicates the available receive buffer; reception is continued. If no buffer of the requested size is available, a receive buffer size of 0 shall be indicated by bufferSizePtr.
	BUFREQ_E_NOT_OK Connection has been rejected; reception is aborted. bufferSizePtr remains unchanged.
	BUFREQ_E_OVFL No buffer of the required length can be provided; reception is aborted. bufferSizePtr remains unchanged.
Functional Description	
Receive indication callback for TP. This function is called at the start of receiving an N-SDU. The N-SDU might be fragmented into multiple N-PDUs or might consist of a single N-PDU.	
Particularities and Limitations	
-	
Call context	

> TASK|ISR2 > This function is Synchronous > This function is Non-Reentrant

Table 4-24 DoIP_SoAdTpStartOfReception

4.4.5 DoIP_SoAdTpRxIndication

Prototype	
`void DoIP_SoAdTpRxIndication (PduIdType id, Std_ReturnType result)`	
Parameter	
id [in]	Identification of the received TP Pdu.
result [in]	Result of the reception. Range: NTFRSLT_OK, NTFRSLT_E_NOT_OK.
Return code	
void	none
Functional Description	
Receive indication callback for TP. Called after an TP Pdu has been received via the TP API, the result indicates whether the transmission was successful or not.	
Particularities and Limitations	
-	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Reentrant	

Table 4-25 DoIP_SoAdTpRxIndication

4.4.6 DoIP_SoAdIfRxIndication

Prototype	
`void DoIP_SoAdIfRxIndication (PduIdType RxPduId, const PduInfoType *PduInfoPtr)`	
Parameter	
RxPduId [in]	Id of the received IF Pdu.
PduInfoPtr [in]	Contains the length (SduLength) of the received IF Pdu and a pointer to a buffer (SduDataPtr) containing the IF Pdu.
Return code	
void	none
Functional Description	
Receive indication callback for IF. Indication of a received IF Pdu from a lower layer communication interface module.	

Particularities and Limitations
-
Call context
> TASK|ISR2 > This function is Synchronous > This function is Non-Reentrant for the same ID but Reentrant for different IDs.

Table 4-26 DoIP_SoAdIfRxIndication

4.4.7 DoIP_SoAdIfTxConfirmation

Prototype	
`void DoIP_SoAdIfTxConfirmation (PduIdType TxPduId)`	
Parameter	
TxPduId [in]	Id of the IF Pdu that has been transmitted.
Return code	
void	none
Functional Description	
Transmission confirmation callback for IF. The lower layer communication interface module confirms the transmission of an IF Pdu.	
Particularities and Limitations	
No functionality is implemented. -	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Reentrant	

Table 4-27 DoIP_SoAdIfTxConfirmation

4.4.8 DoIP_SoConModeChg

Prototype	
`void DoIP_SoConModeChg (SoAd_SoConIdType SoConId, SoAd_SoConModeType Mode)`	
Parameter	
SoConId [in]	Socket connection index specifying the socket connection with the mode change.
Mode [in]	New mode. Range: SOAD_SOCON_*.
Return code	
void	none
Functional Description	
Socket state change callback.	

Notification about a SoAd socket connection state change, e.g. socket connection gets online.
Particularities and Limitations
-
Call context
> TASK|ISR2 > This function is Synchronous > This function is Non-Reentrant for the same ID but Reentrant for different IDs.

Table 4-28 DoIP_SoConModeChg

4.4.9 DoIP_LocalIpAddrAssignmentChg

Prototype	
`void DoIP_LocalIpAddrAssignmentChg (SoAd_SoConIdType SoConId, SoAd_IpAddrStateType State)`	
Parameter	
SoConId [in]	Socket connection index specifying the socket connection where the IP address assigment has changed.
State [in]	State of IP address assignment. Range: SOAD_IPADDR_STATE_*.
Return code	
void	none
Functional Description	
IP address assignment change callback. This function gets called by the SoAd if an IP address assignment related to a socket connection changes (i.e. new address assigned or assigned address becomes invalid).	
Particularities and Limitations	
-	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Non-Reentrant	

Table 4-29 DoIP_LocalIpAddrAssignmentChg

4.4.10 DoIP_ShutdownFinished

Prototype	
`void DoIP_ShutdownFinished (void)`	
Parameter	
void	none
Return code	
void	none

Functional Description
Indicates shutdown of SoAd. -
Particularities and Limitations
-
Call context
> TASK|ISR2 > This function is Synchronous > This function is Non-Reentrant

Table 4-30 DoIP_ShutdownFinished

4.4.11 DoIP_DhcpEvent

Prototype	
`void DoIP_DhcpEvent (SoAd_LocalAddrIdType LocalIpAddrId, SoAd_DhcpEventType Event)`	
Parameter	
LocalIpAddrId [in]	IP address identifier.
Event [in]	Indicates the received message type or the message type who will be sent.
Return code	
void	none
Functional Description	
Indicates reception of a DHCP option or that a DHCP message will be sent. -	
Particularities and Limitations	
-	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Non-Reentrant	

Table 4-31 DoIP_DhcpEvent

4.5 Configurable Interfaces

4.5.1 Callout Functions

At its configurable interfaces the DoIP defines callout functions. The parameter list of the callout functions are provided by DoIP. The function name can be configured in the configuration tool. It is the integrator's task to provide the corresponding function definitions. The definitions of the callouts can be adjusted to the system's needs. The DoIP callout function declarations are described in the following tables:

4.5.1.1 <User>_DoIPGetVinCallback

Prototype		
`Std_ReturnType [DoIPGetVinCallbackName] (uint8 *Vin)`		
Parameter		
Vin [in]	Pointer to buffer with a size of exact 17 bytes where the VIN shall be stored.	
Return code		
Std_ReturnType	E_OK	Request is accepted.
	E_NOT_OK	Request is not accepted.
Functional Description		
Retrieves VIN from application.		
Particularities and Limitations		
Call context		
> TASK|ISR2 > This function is Synchronous > This function is Non-Reentrant		

Table 4-32 <User>_DoIPGetVinCallback

4.5.1.2 <User>_DoIPGetGidCallback

Prototype		
`Std_ReturnType [DoIPGetGidCallbackName] (uint8 *GroupId)`		
Parameter		
GroupId [in]	Pointer to buffer with a size of exact 6 bytes where the GID shall be stored.	
Return code		
Std_ReturnType	E_OK	Request is accepted.
	E_NOT_OK	Request is not accepted.
Functional Description		
Retrieves GID from application.		
Particularities and Limitations		
Call context		
> TASK|ISR2 > This function is Synchronous > This function is Non-Reentrant		

Table 4-33 <User>_DoIPGetGidCallback

4.5.1.3 <User>_DoIPTriggerGidSyncCallback

Prototype		
`Std_ReturnType` **`[DoIPTriggerGidSyncCallbackName]`** `(void)`		
Parameter		
void [in]	none	
Return code		
Std_ReturnType	E_OK	GroupIdentifier Synchronization was triggered.
	E_NOT_OK	GroupIdentifier Synchronization could not be triggered so try again next MainFunction.
Functional Description		
Triggers GID synchronization at application.		
Particularities and Limitations		
Call context		
> TASK|ISR2 > This function is Synchronous > This function is Non-Reentrant		

Table 4-34 <User>_DoIPTriggerGidSyncCallback

4.5.1.4 <User>_DoIPGetPowerModeCallback

Prototype		
`Std_ReturnType` **`[DoIPGetPowerModeCallbackName]`** `(DoIP_PowerStateType *PowerStateReady)`		
Parameter		
PowerStateReady [in]	Pointer to buffer where the power mode shall be stored.	
Return code		
Std_ReturnType	E_OK	Request is accepted.
	E_NOT_OK	Request is not accepted.
Functional Description		
Retrieves power mode from application.		
Particularities and Limitations		
Call context		
> TASK|ISR2 > This function is Synchronous > This function is Non-Reentrant		

Table 4-35 <User>_DoIPGetPowerModeCallback

4.5.1.5 <User>_DoIPShutdownFinishedCallback

Prototype		
`void [DoIPShutdownFinishedCallbackName] (void)`		
Parameter		
void [in]	none	
Return code		
void	none	
Functional Description		
Informs upper layer about finished shutdown.		
Particularities and Limitations		
Call context		
> TASK|ISR2 > This function is Synchronous > This function is Non-Reentrant		

Table 4-36 <User>_DoIPShutdownFinishedCallback

4.5.1.6 <User>_DoIPRoutingActivationAuthentication

Prototype		
`Std_ReturnType [DoIPRoutingActivationAuthenticationName] (boolean *Authentified, uint8 *AuthenticationReqData, uint8 *AuthenticationResData)`		
Parameter		
Authentified [in]	Indicates if authentication was successful.	
AuthenticationReqData [in]	Pointer to OEM specific part for authentication of routing activation request. Passed data length is configured by the parameter DoIPRoutingActivationAuthenticationReqLength.	
AuthenticationResData [in]	Pointer to OEM specific part for authentication of routing activation response. Passed data length is configured by the parameter DoIPRoutingActivationAuthenticationResLength.	
Return code		
Std_ReturnType	E_OK	Authentified and AuthenticationResData contain valid Data.
	E_NOT_OK	Authentified and/or AuthenticationResData do not contain valid information.
	DOIP_E_PENDING	Authentication still running. Call next DoIP_MainFunction cycle again.
Functional Description		
Forwards OEM specific part for authentication of received routing activation request to application and retrieves OEM specific part for authentication for routing activation response. Via configuration parameter `DoIPRoutingActivationAuthenticationParamRemAddr` it is possible to add a new parameter (`const SoAd_SockAddrType* RemAddrPtr`) to this function to get the remote		

address of the tester sending the corresponding routing activation request.
Particularities and Limitations
Call context
> TASK|ISR2 > This function is Synchronous > This function is Non-Reentrant

Table 4-37 <User>_DoIPRoutingActivationAuthentication

4.5.1.7 <User>_DoIPRoutingActivationConfirmation

Prototype		
`Std_ReturnType` **`[DoIPRoutingActivationConfirmationName]`** `(boolean *Confirmed, uint8 *ConfirmationReqData, uint8 *ConfirmationResData)`		
Parameter		
Confirmed [in]	Indicates if confirmation was successful.	
ConfirmationReqData [in]	Pointer to OEM specific part for confirmation of routing activation request. Passed data length is configured by the parameter DoIPRoutingActivationConfirmationReqLength.	
ConfirmationResData [in]	Pointer to OEM specific part for authentication of routing activation response. Passed data length is configured by the parameter DoIPRoutingActivationConfirmationResLength.	
Return code		
Std_ReturnType	E_OK	Confirmed and ConfirmationResData contain valid Data.
	E_NOT_OK	Confirmed and/or ConfirmationResData do not contain valid information.
	DOIP_E_PENDING	Confirmation still running. Call next DoIP_MainFunction cycle again.
Functional Description		
Forwards OEM specific part for confirmation of received routing activation request to application and retrieves OEM specific part for confirmation for routing activation response. Via configuration parameter `DoIPRoutingActivationConfirmationParamRemAddr` it is possible to add a new parameter (`const SoAd_SockAddrType* RemAddrPtr`) to this function to get the remote address of the tester sending the corresponding routing activation request.		
Particularities and Limitations		
Call context		
> TASK|ISR2 > This function is Synchronous > This function is Non-Reentrant		

Table 4-38 <User>_DoIPRoutingActivationConfirmation

4.5.1.8 <User>_DoIPOemPayloadRxCallback

Prototype		
`Std_ReturnType [DoIPOemPayloadRxCallbackName] (DoIP_ConnectionIdType ConnectionId, uint16 PayloadType,const PduInfoType *PayloadDataPtr, DoIP_OemPayloadTypeFlagType Flags)`		
Parameter		
ConnectionId [in]	DoIP unique identifier of the connection.	
PayloadType [in]	The payload type that has been received.	
PayloadDataPtr [in]	PDU information structure which contains the length and the payload data of the received message.	
Flags [in]	Flags indicates protocol (TCP/UDP) and routing activation state.	
Return code		
Std_ReturnType	E_OK	Known payload type
	E_NOT_OK	Unknown payload type
Functional Description		
Forwards user data of manufacturer-specific payload types to user.		
Particularities and Limitations		
Call context		
> TASK|ISR2 > This function is Synchronous > This function is Non-Reentrant for the same ID but Reentrant for different IDs.		

Table 4-39 <User>_DoIPOemPayloadRxCallback

4.5.1.9 <User>_DoIPVerifyTargetAddressCallback

Prototype		
`Std_ReturnType [DoIPVerifyTargetAddressCallbackName] (uint16 TargetAddr)`		
Parameter		
TargetAddr [in]	Received logical target address.	
Return code		
Std_ReturnType	E_OK	Target address is accepted.
	E_NOT_OK	Target address is declined.
Functional Description		
Forwards logical target address received within a diagnostic message to user and accepts or declines the target address depending on return value.		
Particularities and Limitations		
Call context		
> TASK		

> This function is Synchronous > This function is Non-Reentrant

Table 4-40 <User>_DoIPVerifyTargetAddressCallback

4.5.1.10 <User>_DoIPVerifyRxPduCallback

Prototype		
`Std_ReturnType [DoIPVerifyRxPduCallbackName] (const SoAd_SockAddrType * LocalAddrPtr, const SoAd_SockAddrType * RemoteAddrPtr, uint16 SourceAddr, uint16 TargetAddr, const PduInfoType * PduInfoPtr)`		
Parameter		
LocalAddrPtr [in]	Pointer to local socket address	
RemoteAddrPtr[in]	Pointer to remote socket address	
SourceAddr[in]	Logical source address	
TargetAddr[in]	Logical target address	
PduInfoPtr[in]	Pointer to PDU	
Return code		
Std_ReturnType	E_OK	PDU reception verification succeeded.
	E_NOT_OK	PDU reception verification failed.
Functional Description		
Verifies a PDU (diagnostic message) reception and indicates if a reception shall be continued or dropped.		
Particularities and Limitations		
Call context		
> TASK > This function is Synchronous > This function is Non-Reentrant		

Table 4-41 <User>_DoIPVerifyRxPduCallback

4.5.1.11 <User>_DoIPUseSecureCommunicationCallback

Prototype		
`boolean [DoIPUseSecureCommunicationCallbackName] (void)`		
Parameter		
void [in]	none	
Return code		
boolean	TRUE	Secure communication required.
	FALSE	Insecure communication is accepted too.
Functional Description		
Decides if secure communication is required or if insecure communication is accepted too.		

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

Table 4-42 <User>_DoIPUseSecureCommunicationCallback

5 Configuration

There is one configuration tool to configure and generate the DoIP.

> DaVinci Configurator

5.1 Configuration Variants

The DoIP supports the configuration variants

> VARIANT-PRE-COMPILE

> VARIANT-POST-BUILD-LOADABLE

> VARIANT-POST-BUILD-SELECTABLE

The configuration classes of the DoIP parameters depend on the supported configuration variants. For their definitions please see the DoIP_bswmd.arxml file.

5.2 Configuration of Post-Build

The configuration of post-build loadable is described in [7].

The configuration of post-build selectable is described in [8].

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

5.3 Configuration Procedure

This chapter gives a short introduction of how to configure DoIP.

5.3.1 Basic functionality

This chapter gives some background information to understand how DoIP must be configured.

5.3.1.1 Callback functions

At least a callback for VIN and Power Mode must be available to run DoIP. Please choose function names for both parameters. Depending on configuration further callback functions may be required.

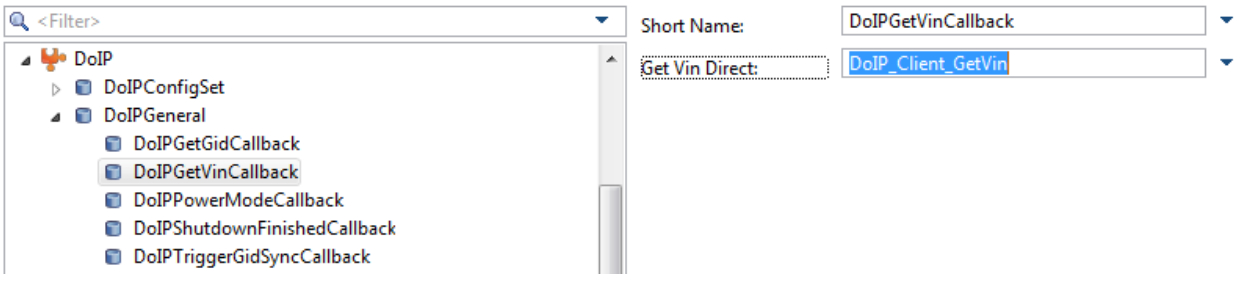

Figure 5-1 Configure callback functions

5.3.1.2 DoIP Interface

A DoIP Interface is a self-contained unit and defines a logical IP interface. The configuration described in the subsequent chapters with respect to the Connections, Channels, Testers and Routing Activations are required per DoIP Interface.

5.3.1.3 TCP/UDP connection configuration (interface to SoAd)

First it is required to configure exactly one UDP connection to handle unicast messages (e.g. Vehicle Identification Request). It is possible to configure multiple UDP connections to handle UDP Requests from different testers at the same time. An example is given in Figure 5-2.

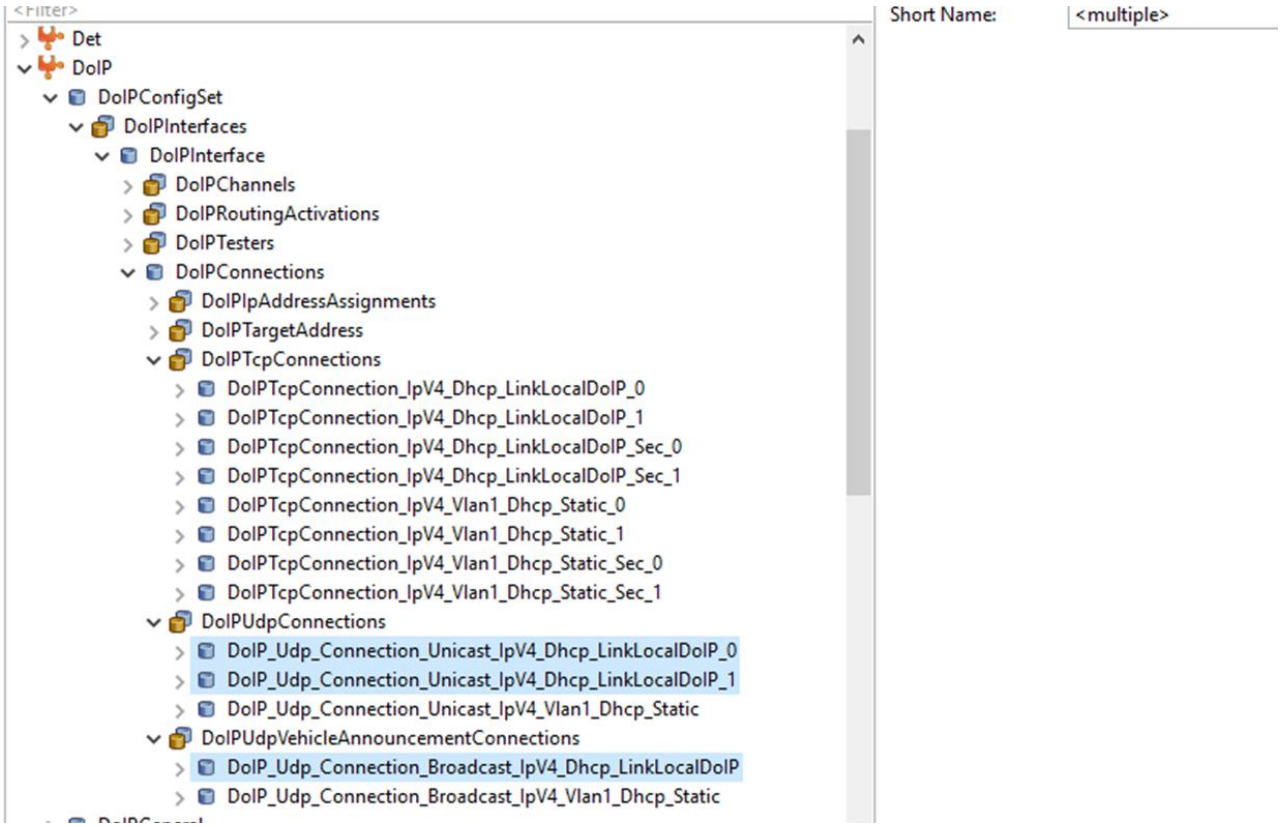

Figure 5-2 UDP connection and UDP Vehicle Announcement connection configuration

After creation of the mentioned connections for UDP, SoAd has to be configured. Please refer to SoAd documentation to get a functional overview about the module.

Create for each UDP connection a corresponding `SoAdPduRoute` and `SoAdSocketRoute`. For the UDP Vehicle Announcement connection a `SoAdPduRoute` is required only (Figure 5-3).

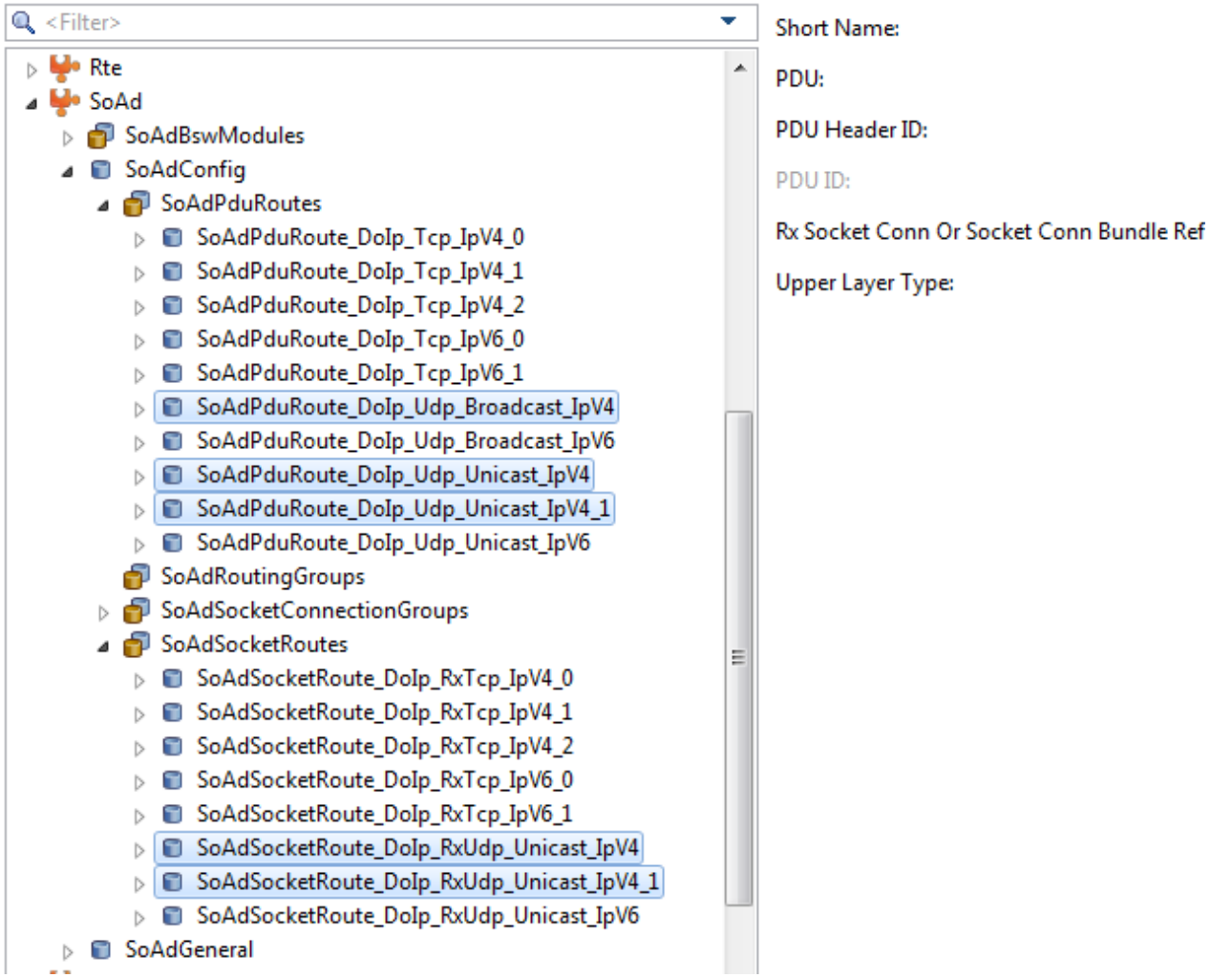

Figure 5-3 UDP connection and UDP Vehicle Announcement connection SoAd counterpart configuration

Both UDP connection types share one `SoAdSocketConnectionGroup` but each connection has an own `SoAdSocketConnection`.

UDP Vehicle Announcement connections have to configure a remote address set to the IPv4 limited broadcast address or IPv6 link-local scope multicast address and a remote port set according to [5] (Figure 5-4).

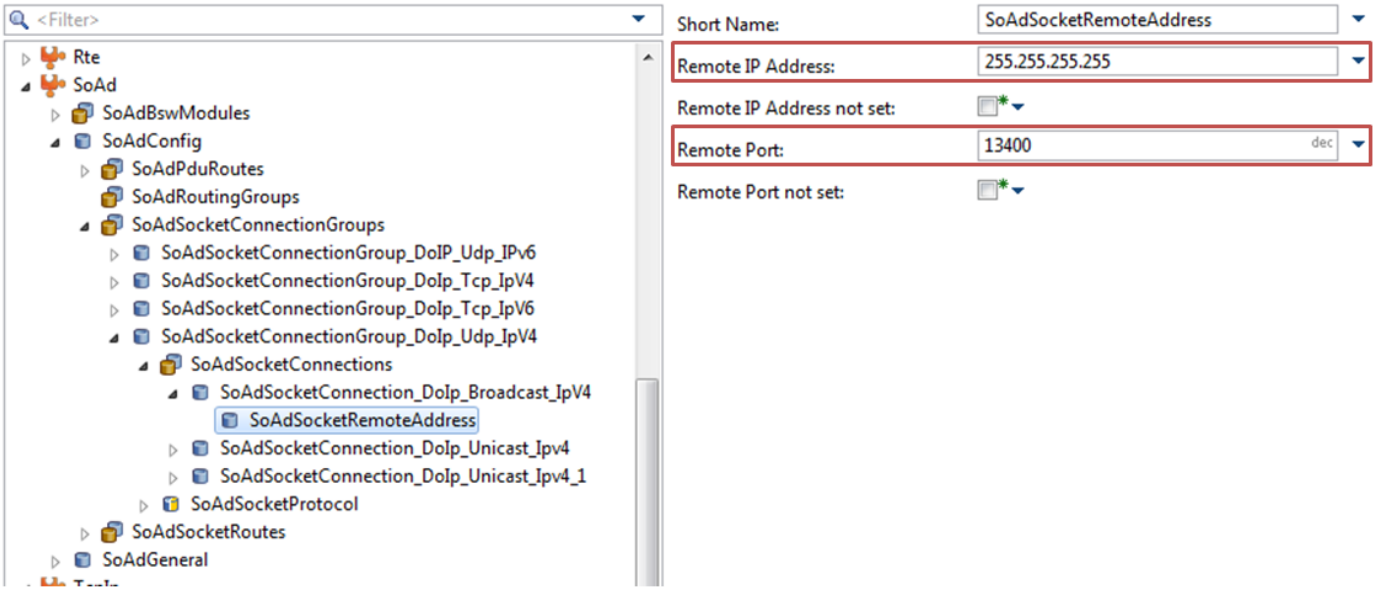

Figure 5-4 UDP Vehicle Announcement connection remote address configuration

UDP connections have to configure a remote address set to wildcard (Figure 5-5).

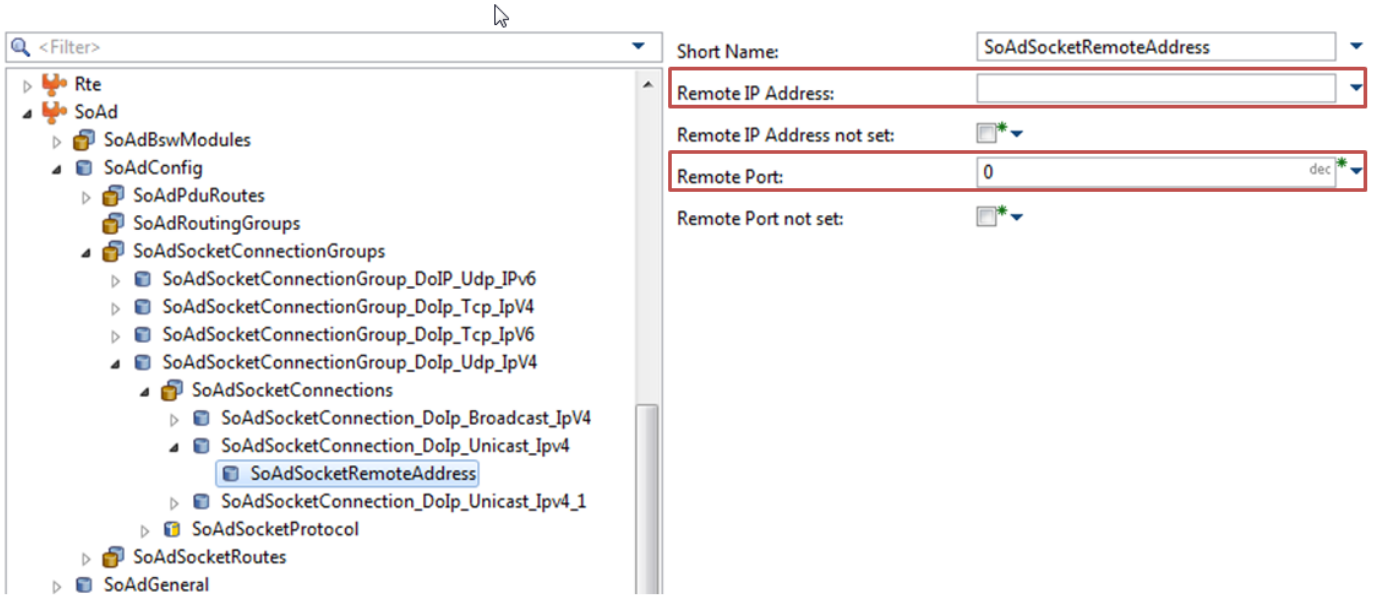

Figure 5-5 UDP connection remote address configuration

After configuration of both types of UDP connections, it is required to configure at least two TCP connections (to reject a second tester within DoIP protocol on the second TCP connection while a first tester is already connected to entity over the first TCP connection). Since DoIP needs at least one tester at least two TCP connections are required. For each further tester that shall be connected parallel an additional TCP connection has to be configured (Figure 5-2).

Similar to the UDP connections configure a `SoAdPduRoute` and a `SoAdSocketRoute` for each TCP connection in SoAd. All TCP connections share one

SoAdSocketConnectionGroup but have separate SoAdSocketConnections. The remote address is set to wildcard.

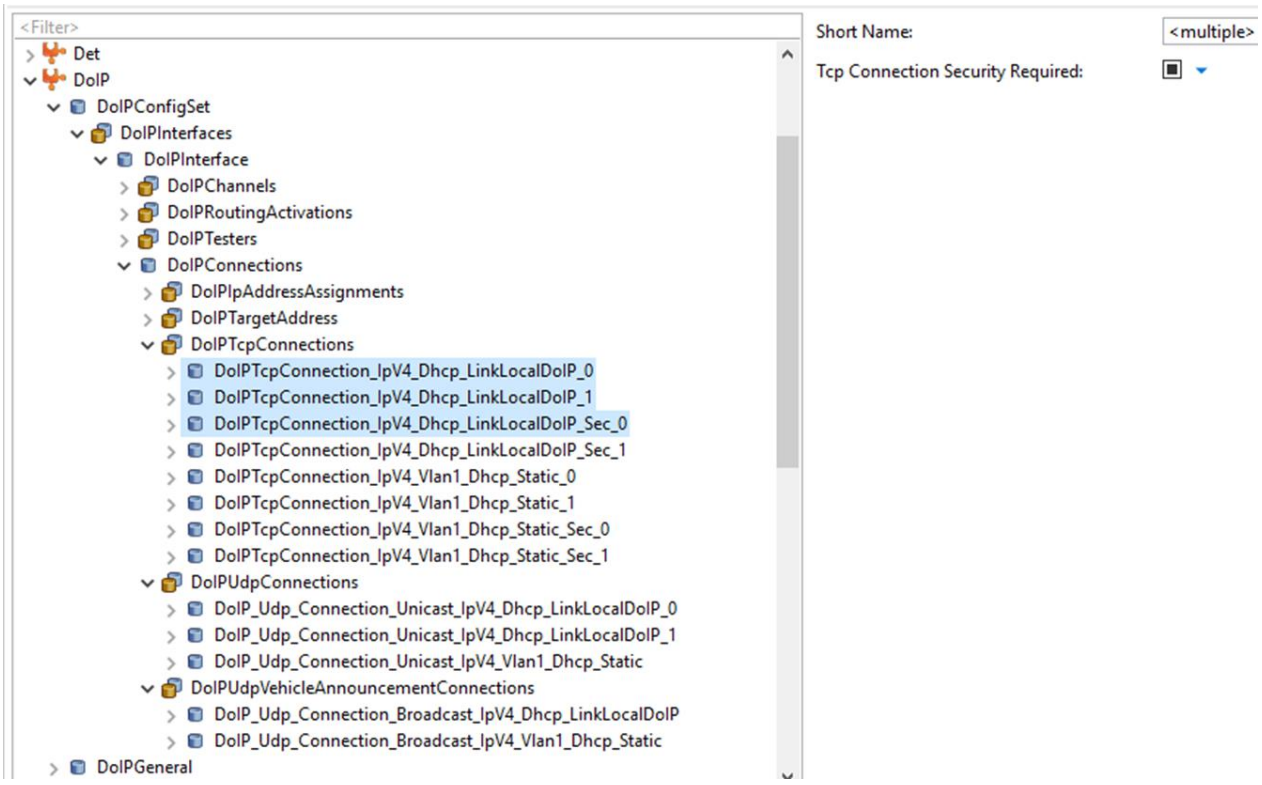

Figure 5-6 TCP connection configuration

5.3.1.4 Channel configuration (interface to PduR)

DoIP defines a diagnostic message to send and receive diagnostic data. A diagnostic message contains a “logical source address” and a “logical target address”. It depends on transmission direction (tester to ECU or ECU to tester) whether the source or target logical address matches the tester address or ECU address.

In configuration an expected tester is described with a DoIPTester container. An ECU address (e.g. entity itself or CAN node behind a gateway) is described with a DoIPTargetAddress container. To send and receive diagnostic messages, DoIPTester and DoIPTargetAddress have to be mapped to each other. The mapping is configured in a DoIPChannel as described in Figure 5-7.

There is no fix mapping of TCP connection to DoIPChannel since mapping is done dynamically at runtime: After establishing a TCP connection (used resource depends on already connected or closing TCP connections) tester sends a routing activation request to activate its address on that connection (mapping of DoIPTester to TCP connection in DoIP module now possible). Afterwards on reception of a diagnostic message DoIP can identify the corresponding DoIPChannel via “logical target address” received within the diagnostic message header.

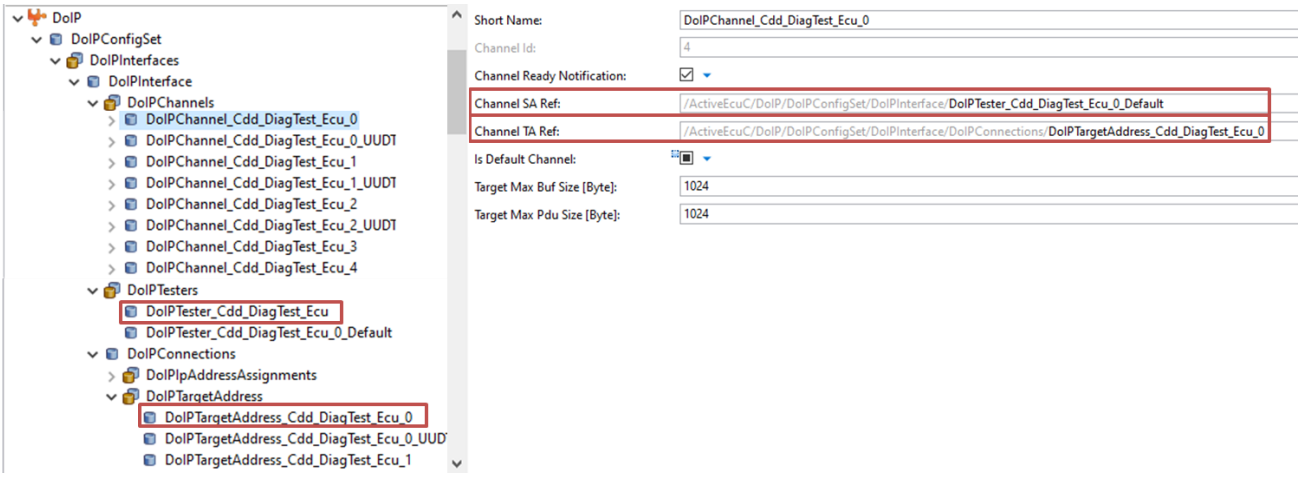

Figure 5-7 Channel configuration

5.3.1.4.1 UUDT configuration

A channel can be configured to be used for UUDT or “normal” diagnostic communication. UUDT communication is specified for transmission only.

While “normal” diagnostic communication is done over TP-API to PduR, IF-API is used in case of UUDT.

To configure a channel for UUDT set parameter “DoIPPduType” to “DoIP_IFPDU”.as mentioned in Figure 5-8.

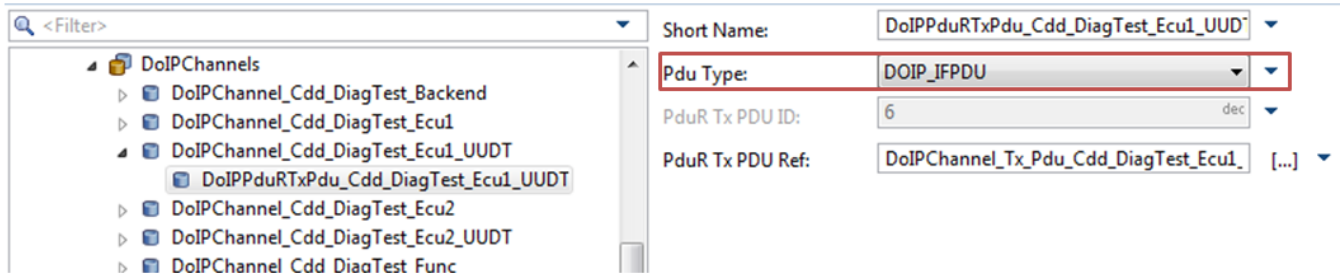

Figure 5-8 Channel UUDT configuration

5.3.1.5 Tester and Routing Activation configuration

Each tester is identified by a logical address (Figure 5-9). Additionally Routing Activation configurations have to be referenced by each tester.

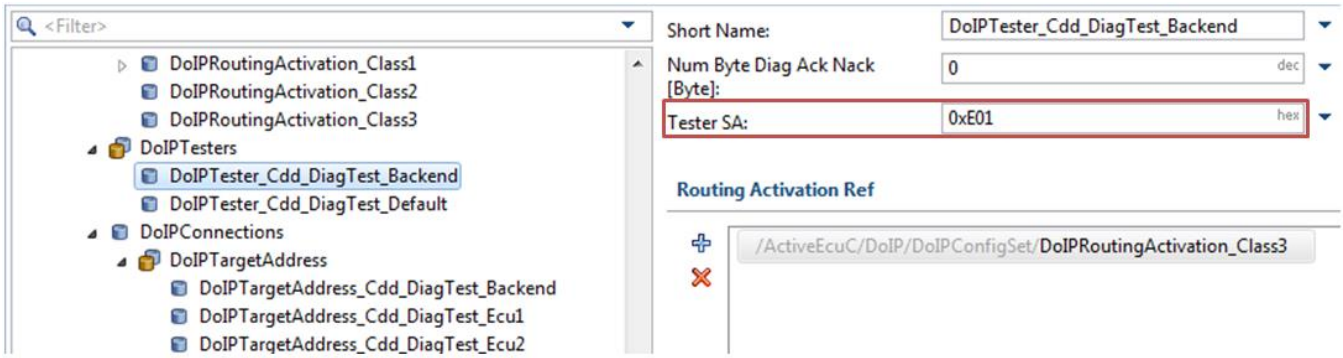

Figure 5-9 Tester logical address

A Routing Activation is identified by a Routing Activation Number (Figure 5-10) which is received as “Activation type” (refer to Table 22 and 23 in [5]) in a Routing Activation Request message.

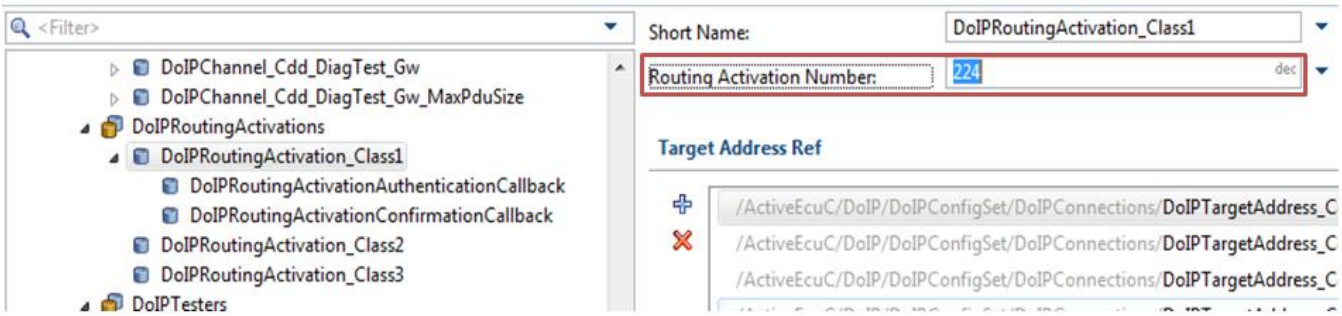

Figure 5-10 Routing Activation Number

If routing activation was successful all target addresses referenced by the Routing Activation (Figure 5-11) are accessible via diagnostic messages.

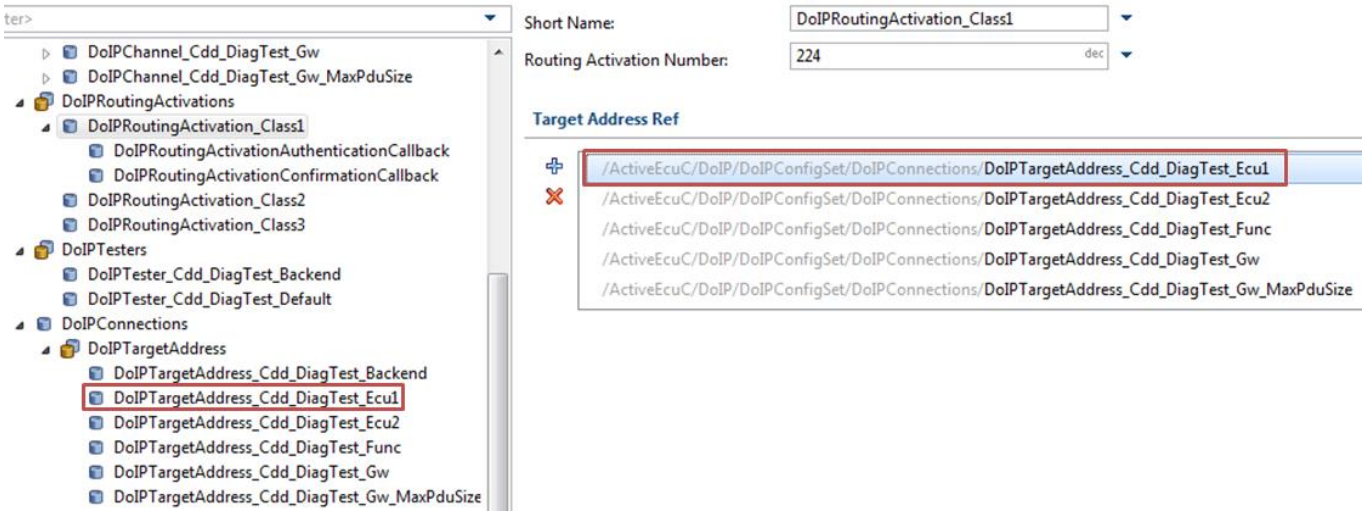

Figure 5-11 Target Address reference in Routing Activation

The “OEM specific” part of Routing Activation Request message (refer to Table 22 in [5]) can be retrieved and set by callbacks configured on a Routing Activation container (Figure 5-12).

Figure 5-12 Routing Activation Authentication/Confirmation callback

The “OEM specific” part can be separated in Authentication and Confirmation part of Routing Activation Request and Response. Figure 5-13 shows how length parameters are interpreted.

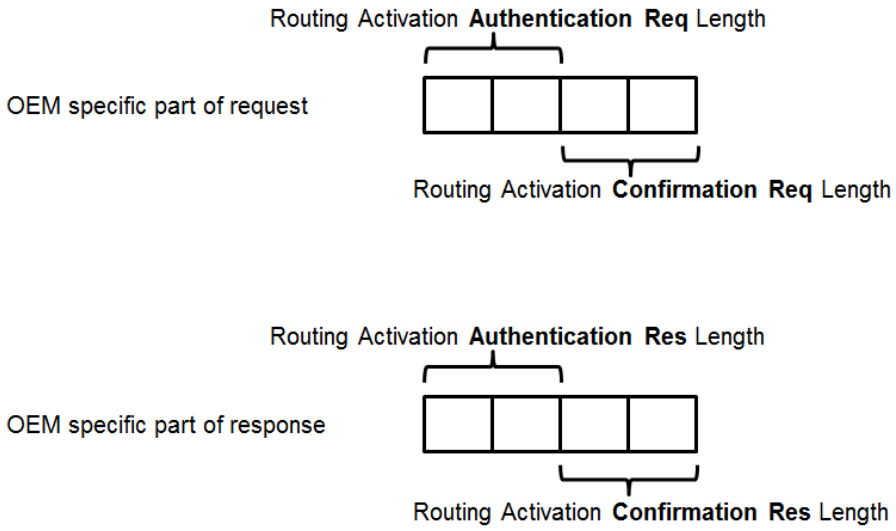

Figure 5-13 Routing Activation Authentication/Confirmation length parameter interpretation

5.3.1.6 Activation Line

The Activation Line can be used to enable or disable diagnostic communication. Its usage can be en-/disabled per DoIP Interface by configuration (parameter `DoIPInterfaceActLineCtrl`). If the Activation Line is active DoIP requests the IP address assignment process on all DoIP dependent local addresses if control by DoIP is configured. If Activation Line is inactive DoIP releases the corresponding IP address assignments.

To switch the Activation Line state call `DoIP_ActivationLineSwitch()` described in 4.2.15. Set parameter `active` to `FALSE` or `TRUE` to indicate if Activation Line is inactive (diagnostic communication disabled) or active (diagnostic communication enabled).

For DoIP Interfaces with disabled Activation Line control, the activation line is always considered as active.

5.3.2 Extended functionality

5.3.2.1 TcpTxQueue

The TcpTxQueue is used to store transmission requests which cannot be handled instantly. All DoIP TCP messages will be stored in this Queue.

For normal operation the queue size must be at least 2 to handle a “Diagnostic Message Acknowledgement” and a corresponding “Diagnostic Message” as a response to a previous received “Diagnostic Message” from tester.

In case of functional requests a DoIP gateway may receive all responses to a request at the same time. To store all responses in the TcpTxQueue the size must be equal to the number of all responses plus 1 (to store the “Diagnostic Message Acknowledgement” from gateway).

To configure the size of TcpTxQueue refer to Figure 5-14.

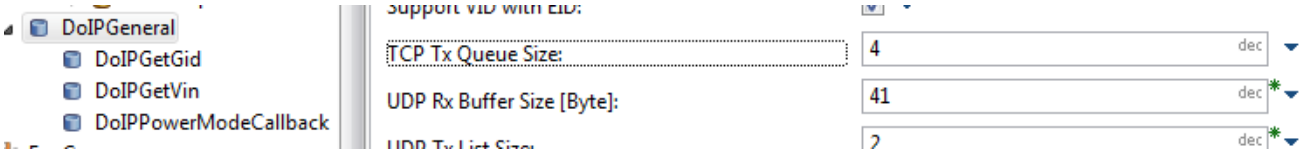

Figure 5-14 Configure TcpTxQueue

If a Vector SoAd is used the parameter

`SoAd/SoAdConfig/SoAdSocketConnectionGroup/SoAdSocketConnection/SoAdTcpTxQueueSize`

should be adapted, too. This queue handles transmission requests and their confirmations. In case of DoIP multiple and short messages may be sent in a short time. Therefore a bigger queue size is required. The size of DoIP TcpTxQueue can be used as approximate value.

To find this parameter the DaVinci Configurator “Find” view can be used.

5.3.2.2 UdpTxList

The UdpTxListSize parameter defines the number of UDP transmission requests the DoIP module can handle at the same time. All types of UDP messages will be stored in this list. The list is shared by all UDP connections. All initial Vehicle Announcements of a local address are handled within one list entry.

To configure the “UdpTxListSize” refer to Figure 5-15.

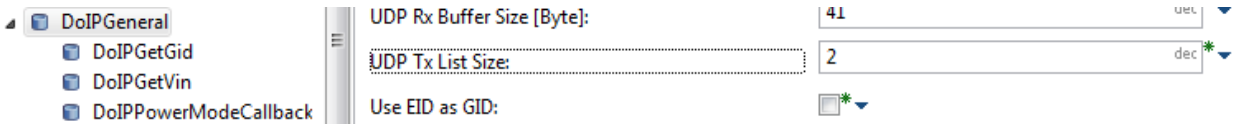

Figure 5-15 Configure UdpTxListSize

5.3.2.3 PDU Size Routing

The PDU Size Routing feature implements a routing of diagnostic data to same target address dependent on message size.

This feature must be enabled in configuration (Figure 5-16) and at runtime via call to DoIP_EnablePduSizeRouting (disable via DoIP_DisablePduSizeRouting).

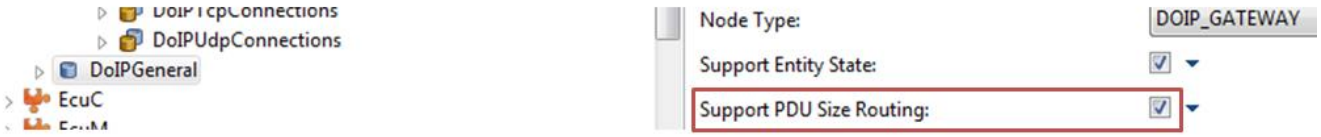

Figure 5-16 PDU Size Routing – enable support

Configure multiple channels with same target address value (refer to referenced DoIPTargetAddress) but different maximum PDU sizes (Figure 5-17 and Figure 5-18).

In the example given in the mentioned figures a reception of diagnostic user data with length <= 200 bytes are routed to DoIPTargetAddress_Cdd_DiagTest**_Gw_1**. On reception of a length > 200 bytes and length <= 256 bytes user data are routed to

DoIPTargetAddress_Cdd_DiagTest**_Gw_1_MaxPduSize**. If length of user data exceeds 256 bytes a “message to large” negative acknowledge is sent and user data are not routed.

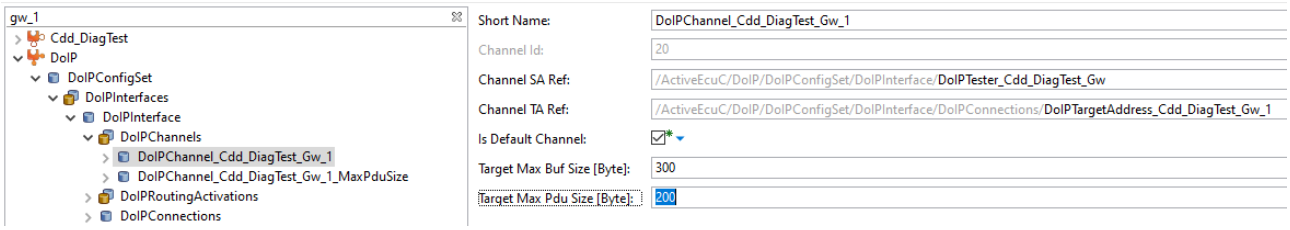

Figure 5-17 PDU Size Routing – target address smaller size and default channel

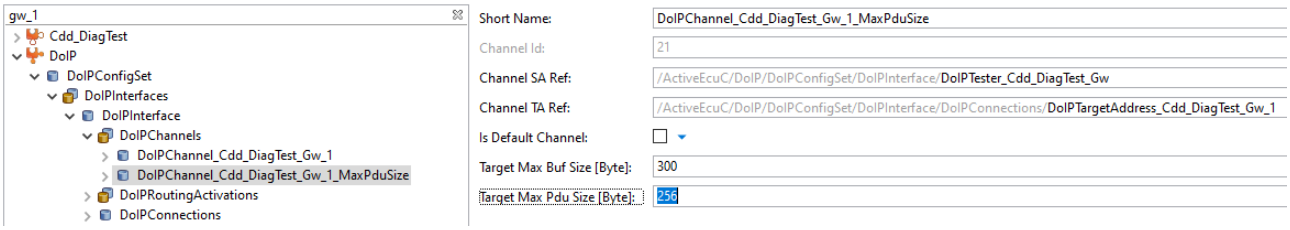

Figure 5-18 PDU Size Routing – target address max size

Exactly one of the channels with same logical target address must be configured to default channel which is used if feature is disabled (Figure 5-17).

5.3.2.4 Default Tester

A channel is restricted to exact one tester i.e. one logical tester address.

To support any logical tester address on a channel a “default tester” i.e. default logical tester address is implemented. Set logical tester address to 0xFFFF to configure a default tester (Figure 5-19).

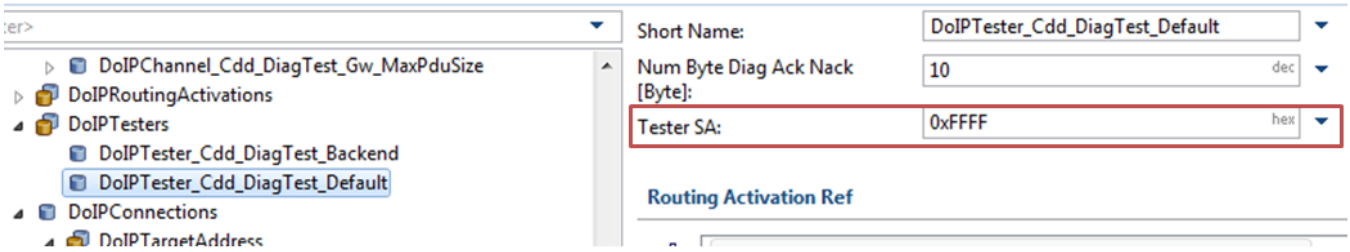

Figure 5-19 Default tester configuration

On reception of a routing activation request with unknown tester address (no other tester with this logical address is configured) a default tester would adopt this address as long as connection is active and act like the address would be configured. Please note that exactly one default tester can be configured currently.

5.3.2.5 Multiple local IP addresses per interface

AUTOSAR assumes that different local IP addresses (e.g. on different VLANs) are configured on different DoIP interfaces. If multiple local IP addresses were configured on the same interface, the IP address assignments would be controlled by DoIP together.

Vector DoIP supports multiple local IP addresses per interface and allows to control the local IP address assignment separately (refer to 5.3.2.11).

Since each local IP address is interpreted as separate entity at least n*2 TCP connections and at least n*1 UDP connections are required for n local IP addresses.

An Alive Check is performed on each local IP address separately in case all TCP_DATA sockets of a local IP address are already activated. Since the tester pool is shared by all local IP addresses, an Alive Check caused by multiple connections to the same tester is performed on all connections independent of the local IP address.

The maximum open sockets field inside the UDP DoIP Entity Status Response message contains the number of all TCP_DATA sockets which are mapped to a local IP address excluding the reserved socket.

5.3.2.5.1 Multiple local IPv4 addresses

According to [5] a AutoIP (i.e. link-local) IP address or a DHCP address can be assigned for IPv4, depending on which address is assigned first.

To support this behavior both IP address assignments must be configured. For IPv4 both assignments can be assigned to one local address (Figure 5-20).

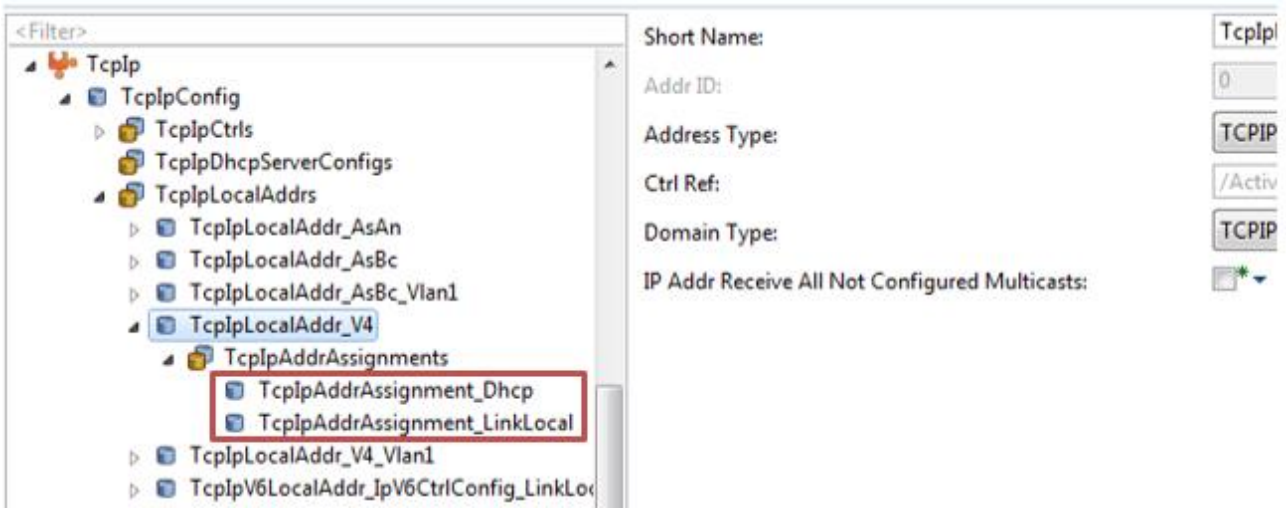

Figure 5-20 Multiple local IPv4 address configuration

For IPv4 multiple local IP addresses on one controller/VLAN means that multiple IP address assignments can be configured but there is just one IP address active at the same time.

5.3.2.5.2 Multiple local IPv6 addresses

According to [5] DoIP shall support multiple local IPv6 addresses (e.g. link-local, DHCP) in parallel.

The configuration is different to IPv4. In case of IPv4 only one local address is configured but multiple assignments can be assigned to this local address (refer to 5.3.2.5.1). For IPv6 each assignment type is assigned to a separate local address (Figure 5-21). Therefore, each IPv6 address must be configured as separate entity in DoIP as described above.

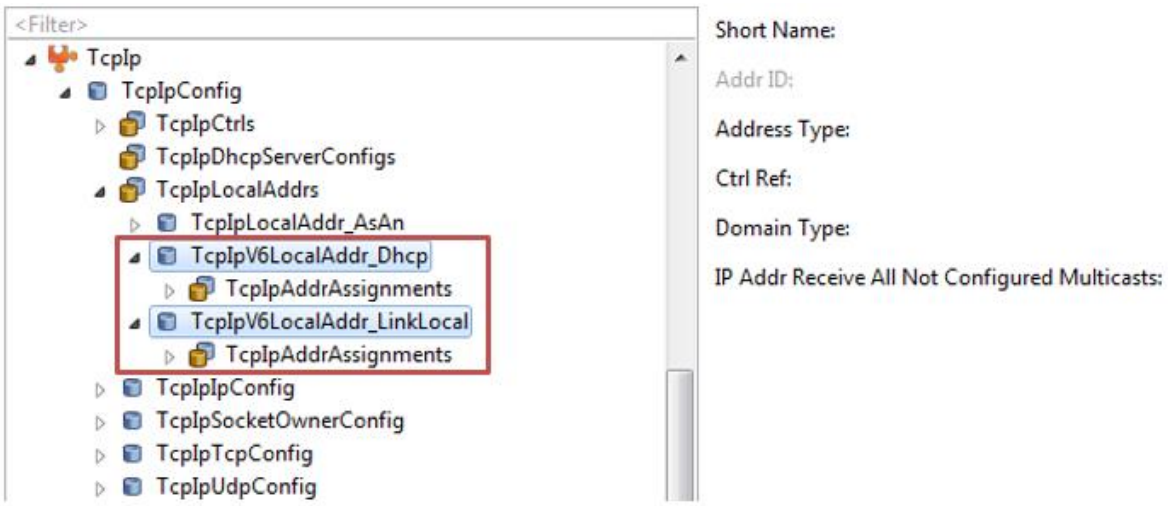

Figure 5-21 Multiple local IPv6 address configuration

5.3.2.6 Shutdown mechanism

Vector DoIP supports a shutdown mechanism to prepare ECU and tester for an application/bootloader context transition. Therefore, all open TCP sockets will be closed when calling `DoIP_Shutdown()` and DoIP module is placed into a specific shutdown state. The shutdown of DoIP is internally performed by calling `SoAd_Shutdown()` which leads to a shutdown of the entire module.

It is possible to configure an optional callback which is called when shutdown is finished. Although this callback is optional it is recommended to configure it to know when to perform ECU reset. It is also possible to poll module state via multiple calls of `DoIP_Shutdown()`.

If sockets cannot be closed since tester does not acknowledge closing (i.e. no “FIN” flag sent) a timeout can be configured to shutdown module after timer expired.

This feature is only available if Vector SoAd is used.

To configure this feature please refer to Figure 5-22 and to Figure 5-23 for callback configuration. SoAd configuration will be adapted via validation rules in configuration tool.

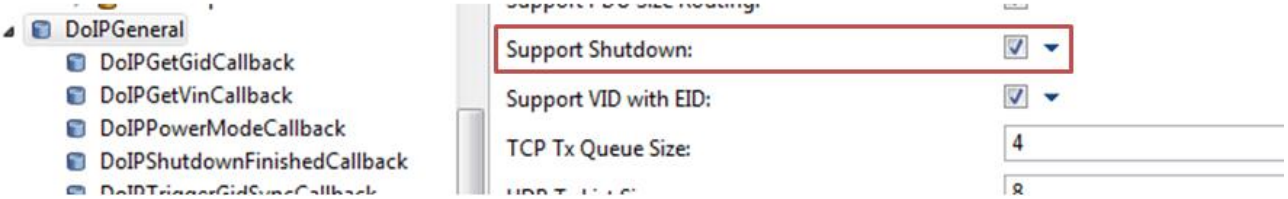

Figure 5-22 Configuration example for DoIP Shutdown

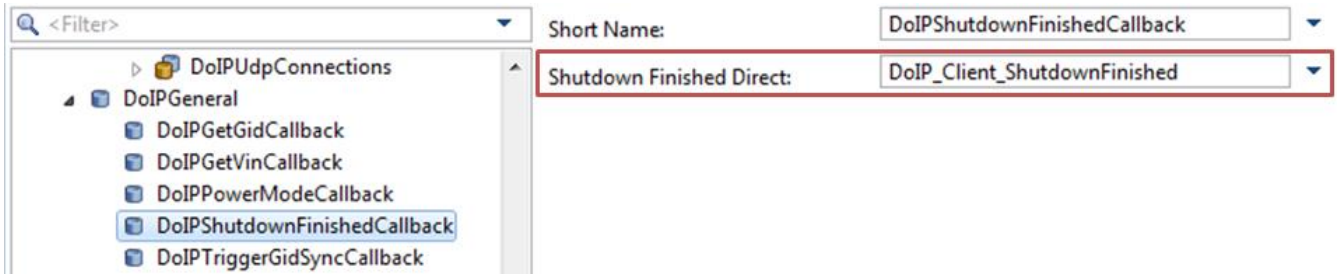

Figure 5-23 Configuration example for DoIP Shutdown callback

Caution
Be aware that the shutdown also affects the other socket connections configured in the Socket Adaptor. This may cause other use-cases not to work afterwards.

5.3.2.7 OEM specific payload type

According to [5] DoIP supports manufacturer-specific payload types in range of 0xF000 to 0xFFFF but [1] does not provide an API to receive or transmit these payload types.

Vector supports both transmission and reception of OEM specific payload types.

For reception, Vector supports a configurable callback to receive all unknown payload types over TCP and UDP. Via return value (refer to 4.5.1.8 for more information) the user can indicate if the payload type is known or unknown. If the user indicates that the payload type is unknown a generic DoIP header negative acknowledge message is sent as specified in [5] and [1].

To enable OEM specific reception it is required to configure the callback via `DoIPOemPayloadRxCallback`. For reception on TCP connections an additional `DoIPOemPayloadRxBuffer` is required (refer to Figure 5-24). There is at maximum one buffer occupied by a connection at the same time. On UDP connections this is not necessary since the received data is directly forwarded and no buffering is required.

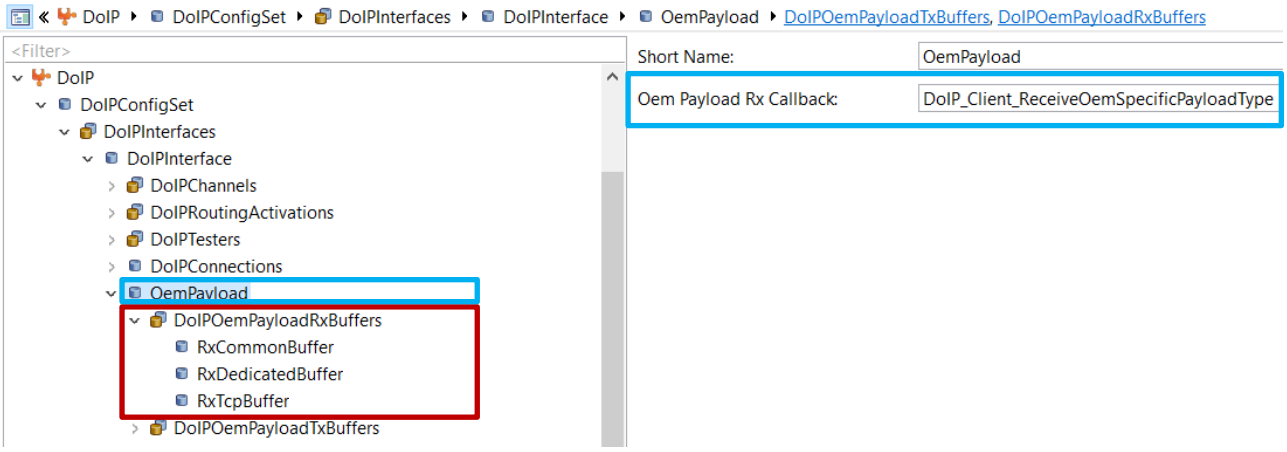

Figure 5-24 OEM specific payload type reception configuration

For transmission, the API `DoIP_TransmitOemSpecificPayloadType()` is provided as defined in chapter 4.2.9.

To be able to transmit OEM specific data on TCP, UDP or UDP Vehicle Announcement connections `DoIPOemPayloadTxBuffer` must be configured as shown in Figure 5-25. Please note that on generation of the buffers an additional length of 8 bytes is added to the configured `DoIPOemPayloadTxBufferSize` since the generic header is stored in this buffer as well in case of UDP connections.

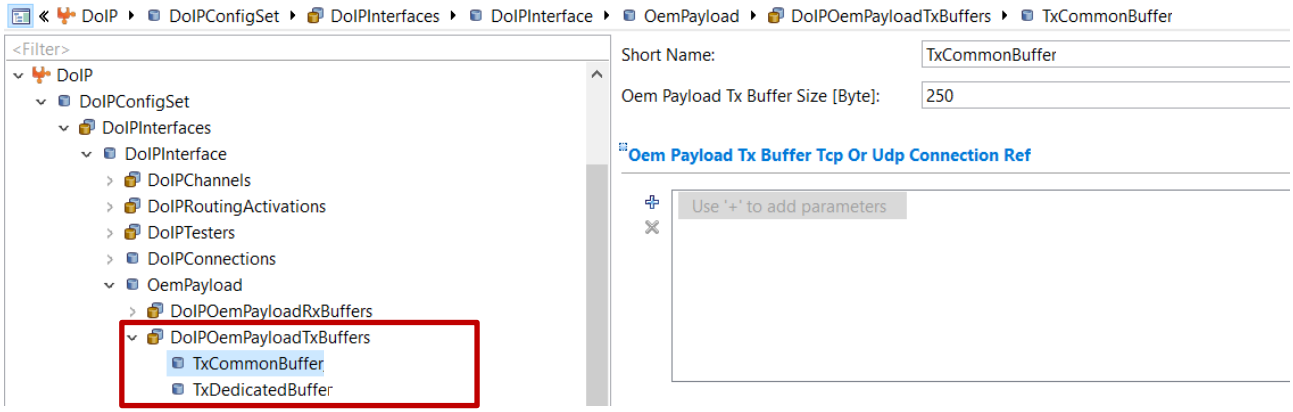

Figure 5-25 OEM specific payload type transmission configuration

For `DoIPOemPayloadRxBuffer` and `DoIPOemPayloadTxBuffer` one or several dedicated connections of an interface may be referenced via `DoIPOemPayloadRxBufferTcpConnectionRef` and `DoIPOemPayloadTxBufferTcpOrUdpConnectionRef`. In case no connections are referenced the buffer is a global buffer (refer to the second buffer shown in Figure 5-26) and can be used for reception or transmission on all connections of the interface. Buffers which reference exactly one connection are so called dedicated buffers and can only be used on the corresponding connection as for example the first buffer shown in Figure 5-26. In case a buffer references several connections, this is a shared (dedicated) buffer which can be used for reception or transmission on these connections (refer to the third buffer shown in Figure 5-26). Based on the buffer configuration, each connection has an own set of buffers which can be used on transmission or reception of OEM specific payload type messages (one buffer is used by one message at the same time). A connection uses dedicated buffers first, shared buffers second and global buffers last. I.e. `DoIP_Con1` shown in Figure 5-26 will first of all try to reserve `RxDedicatedBuffer`, secondly `RxSharedDedicatedBuffer` and `RxGlobalBuffer` last. The buffers are released again when the message has been sent or forwarded to the user completely.

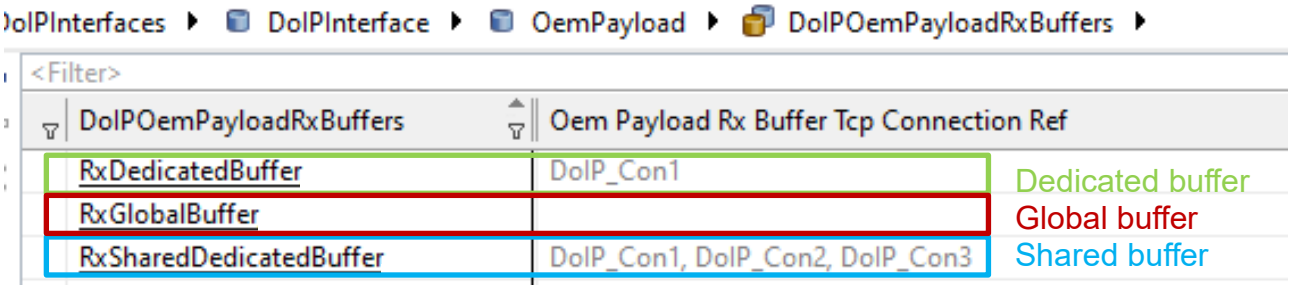

Figure 5-26 OEM specific payload type buffer configuration

Note
DoIP will forward all unknown payload types to user and accepts OEM specific transmission requests with any payload type and is not restricted to the range defined for manufacturer-specific payload types (0xF000 to 0xFFFF).

Note
Be aware that at least one buffer with a minimum length of 1 byte needs to be configured for a connection to enable OEM specific reception/transmission even if no user data is provided.

Caution
In case no suiting buffer can be found on reception (e.g. all buffers are in use or the configured size is not big enough) a generic DoIP header negative acknowledge message is sent.

In case no suiting buffer can be found on transmission the request is rejected.

Caution
In case of OEM specific reception on a UDP connection it has to be considered that the UDP connection must be closed manually if the corresponding `SoAdSocketUdpAliveSupervisionTimeout` is not configured to enable other testers to reuse this connection. This does not apply for UDP Vehicle Announcement connections. Call `DoIP_CloseConnection()` to close UDP connections.

5.3.2.8 Target Address Masking and Verification

5.3.2.8.1 Target Address Masking

Target Address Masking allows receiving diagnostic messages within a value range dependent on configured target address. A bit mask can be configured to mark which bits of received target address has to match the configured target address on a channel to forward the diagnostic data to this channel.

Find an example in Figure 5-27.

Example:
DoIPTargetAddressBitMask set to 0x00FF
→ means first 8 Bit are evaluated on reception only

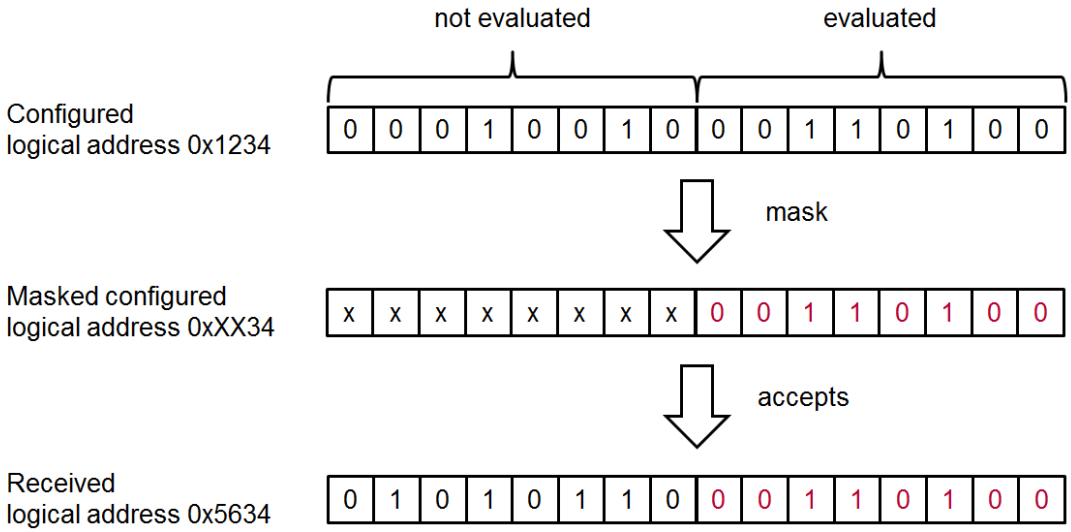

Figure 5-27 Target Address Masking mechanism

The Masking can be configured on each channel (i.e. corresponding target address configuration) separately (Figure 5-28).

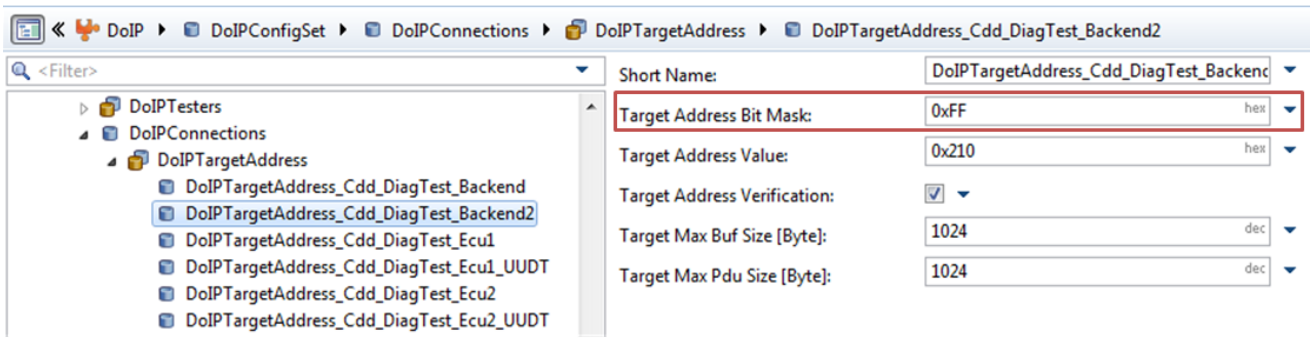

Figure 5-28 Target Address Masking configuration

Note
Overlapping masked target address ranges cannot be used to receive diagnostic data on multiple channels. Exact one matching channel is chosen.

Note
Please note that on transmission the configured target address is used and not the target address which was used on reception. This leads to the side effect that the logical address on reception and on transmission may look different.

5.3.2.8.2 Target Address Verification

Additionally a callback (4.5.1.9) can be configured which is called by DoIP before forwarding diagnostic data after channel has been chosen according to the configured target address (and bit mask if configured). The received target address can be checked within this callback and accepted or declined which is indicated by return value.

Usage of verification callback can be configured optionally on each channel (i.e. corresponding target address configuration) as described in Figure 5-29. The callback name can be configured freely (Figure 5-30).

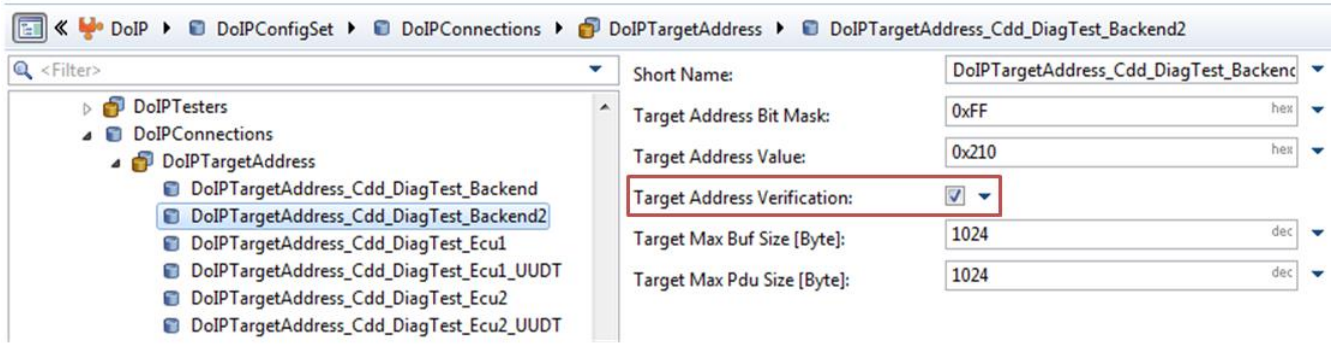

Figure 5-29 Target Address Verification target address configuration

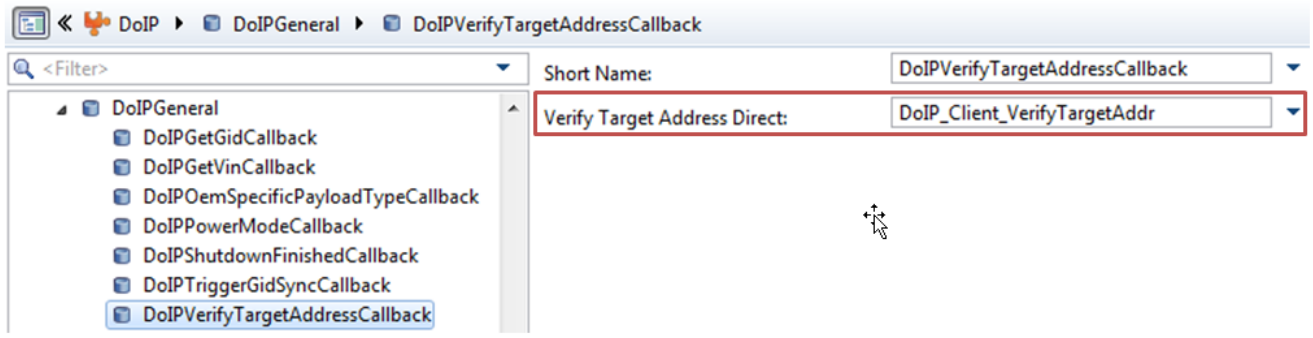

Figure 5-30 Target Address Verification callback configuration

5.3.2.9 Optimized TP transmission

The SoAd according to AUTOSAR can handle maximum one TP transmission per main function on a PDU. Since one Tx PDU represents one TCP connection only one DoIP

message can be sent per main function to the connected tester. The Vector SoAd supports a feature to handle a TP transmission in context of `SoAd_TpTransmit()`. This feature can be used to transmit multiple DoIP messages to tester instead of one per main function cycle only.

To enable this feature enable the corresponding parameter in SoAd module:

`SoAd/SoAdConfig/SoAdPduRoute/SoAdTxTpOptimized`

Caution
If this feature is enabled entire transmission (i.e. DoIP message copied to TCP Tx buffer) is performed in interrupt context if `DoIP_TpTransmit()` is called in interrupt context.

Additionally to handle entire transmission in `SoAd_TpTransmit()` (i.e. including `SoAd_DoIPTpTxConfirmation()`) enable the following parameter in SoAd module:

`SoAd/SoAdConfig/SoAdSocketConnectionGroup/SoAdSocketProtocol/SoAdSocketTcp/SoAdSocketTcpImmediateTpTxConfirmation`

Also consider the following parameter to make sure that a suitable number of transmissions can be handled by the TCP queue of SoAd:

`SoAd/SoAdConfig/SoAdSocketConnectionGroup/SoAdSocketConnection/SoAdTcpTxQueueSize`

If TCP queue of SoAd is not sufficient transmissions are delayed but not discarded.

5.3.2.10 PDU reception verification

DoIP supports PDU reception verification for diagnostic messages. On reception of a diagnostic message DoIP calls a 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. DoIP Logical Source Address
4. DoIP Logical Target Address
5. PDU data

This feature can be used to implement a firewall on DoIP level. In case callback (chapter 4.5.1.10) is successful DoIP forwards the PDU as configured. In case callback fails DoIP drops the PDU and continues with the reception of PDUs received afterwards. In the latter case DoIP sends a diagnostic message positive acknowledgement.

Figure Figure 5-31 shows how to configure the name of the callback and the maximum amount of PDU data which are forwarded via the callback.

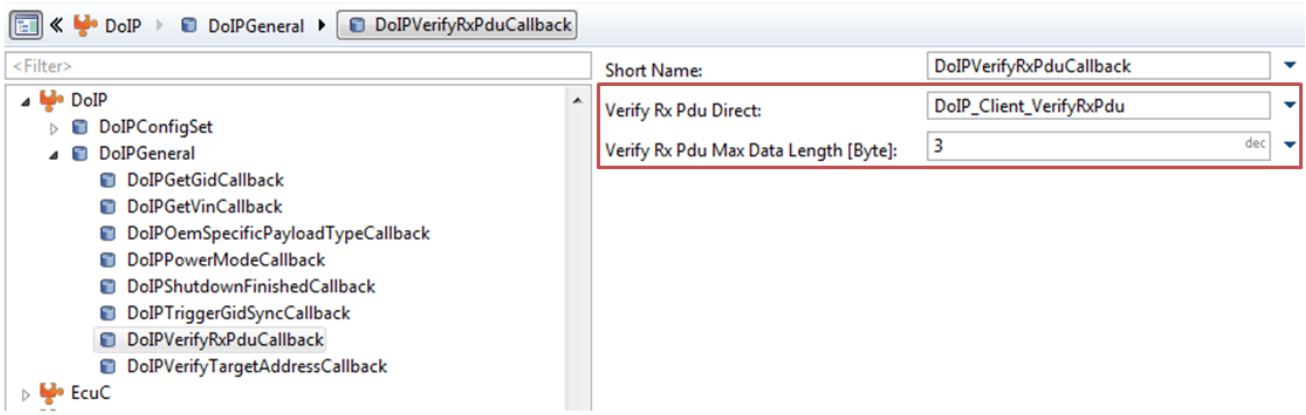

Figure 5-31 PDU reception verification callback configuration

5.3.2.11 Disable control of IP address assignment

According to [1] IP address assignment is requested for every connection with enabled `DoIPRequestAddressAssignment` by DoIP. This applies to all address assignments of the related IP address. For a higher flexibility, the MICROSAR Classic DoIP uses an abstraction of each DoIP related TcpIp IP address assignment. The control of each IP address assignment can be en-/disabled individually as shown in Figure 5-32. Anyway, the parameter `DoIPRequestAddressAssignment` is provided by the MICROSAR Classic DoIP. To ensure consistency with the behavior described in [1], the following mapping rules should be applied:

- `DoIPRequestAddressAssignment` should be set to `FALSE`, in case the IP address assignment control is disabled for at least one IP address assignment related to the local IP address of the connection
- `DoIPRequestAddressAssignment` should be set to `TRUE`, in case the IP address assignment control is enabled for all IP address assignments related to the local IP address of the connection

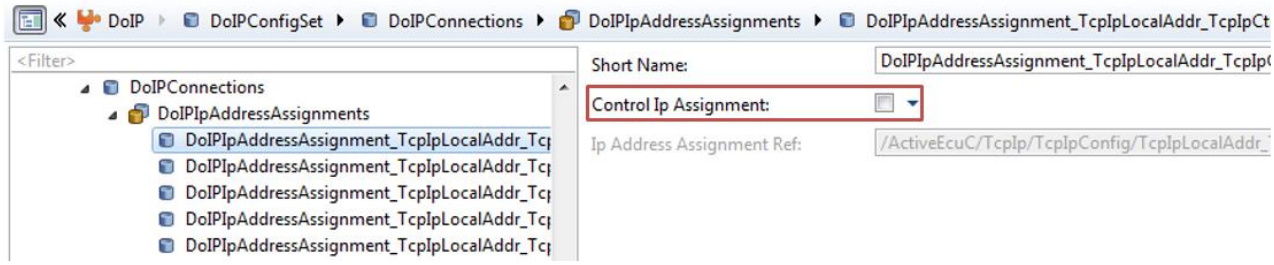

Figure 5-32 Disable control IP address assignment

Note
In case control of IP address assignment is disabled but the assignment is not configured to start automatically another user must request this IP address assignment to start communication over DoIP on this IP address.

5.3.2.12 DHCP option field to be used for target IP addresses of DoIP

This feature will send additional vehicle announcement messages to the IP address list received by the DHCP vendor class option for enterprise number 3210.

With the parameter mentioned in Figure 5-33 the feature can be enabled/disabled.

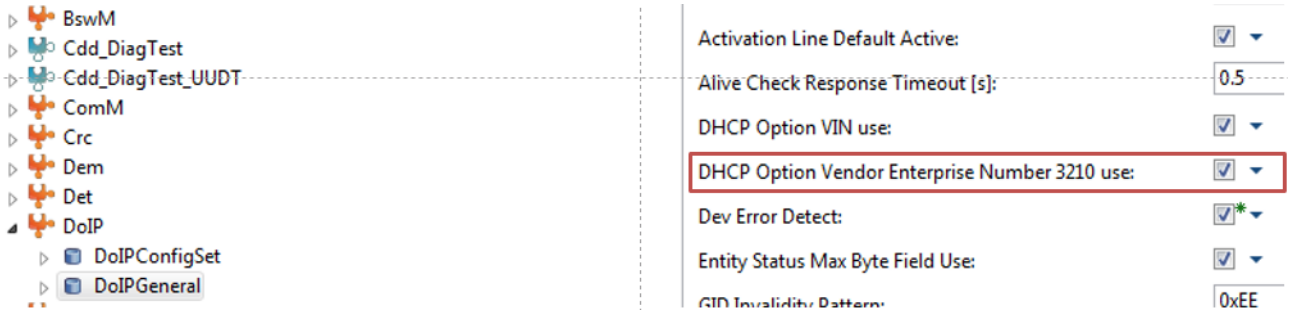

Figure 5-33 DHCP Vendor Class Option feature for Enterprise Number 3210

The feature requires a SoAd callback like mentioned in Figure 5-34.

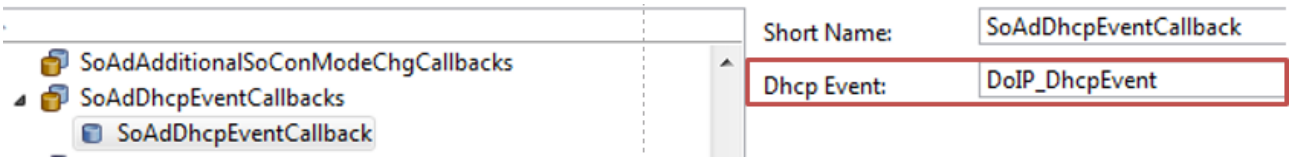

Figure 5-34 DHCP event callback configuration

To receive the DHCP vendor class option the TcpIp has to be configured for DHCPv4 like in Figure 5-35 and for DHCPv6 like in Figure 5-36.

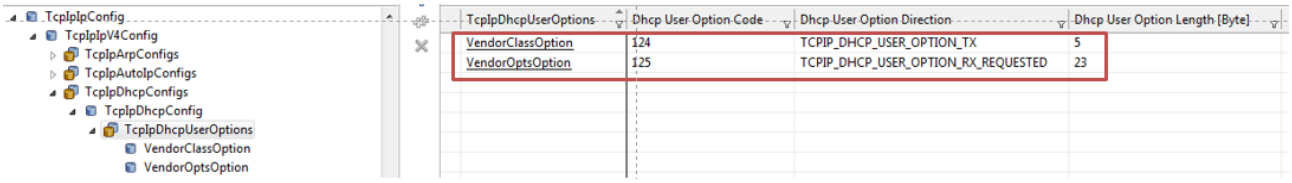

Figure 5-35 DHCPv4 vendor class option configuration

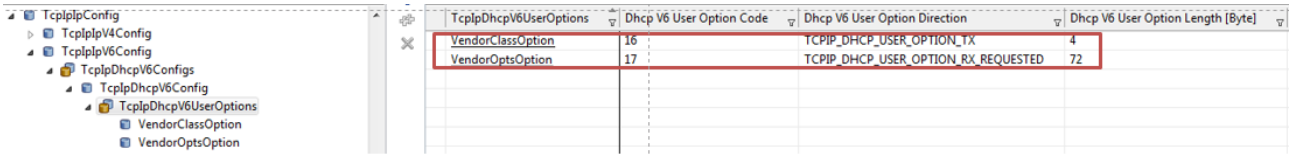

Figure 5-36 DHCPv6 vendor class option configuration

5.3.2.13 Overwrite logical target address

This feature allows overwriting the configured logical target address of a channel during runtime, after initialization of the DoIP module, via the API `DoIP_OverwriteLogicalTargetAddress()`.

As only the logical target address is overwritten, the address masking must be considered when changing the value (see Chapter 5.3.2.8.1 for more details). It is up to the user to ensure that the new logical target address is still unambiguous.

No checks about ongoing transmission/reception procedures are implemented upon overwriting a logical target address. Hence, the user must ensure that it's triggered at the proper time.

An overwritten logical target address is not stored persistently. Therefore, the configured logical target address will be used again after the next initialization of the DoIP module.

To enable this feature, enable the corresponding parameter in the DoIP module:

`/MICROSAR/DoIP/DoIPGeneral/DoIPOverwriteLogicalTargetAddressApi`

Note
Usage of this feature might lead to increased RAM usage and runtime (because internal optimizations when searching a channel can't be used anymore). It is expected that this effect is the greater the more channels are configured.

5.3.2.14 Soft activation line state change

If the Interface activation line inactive timeout is configured to a value unequal to zero, this feature is enabled. The configured timeout is used to close the socket connections gracefully with a TCP FIN while the timeout is not yet reached and close the socket connections immediately with a TCP RST if the timeout is reached. The timeout starts on reception of an activation line inactive request.

Note
If the ECU closes the socket connection with a FIN the TCP socket stays in TIME_WAIT state for 2x MSL until a new connection can be established.

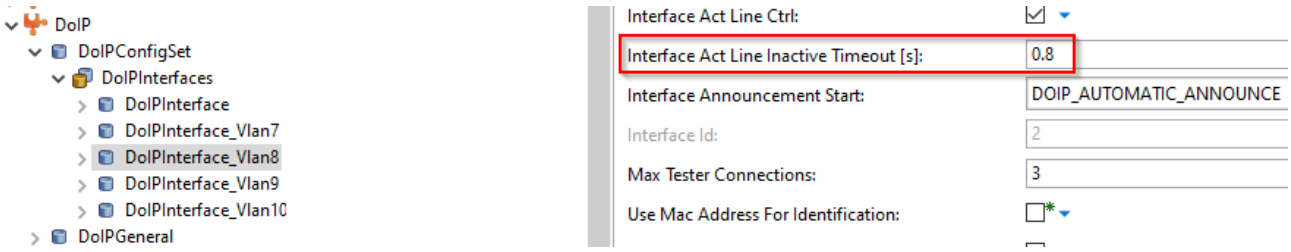

Figure 5-37 Interface activation line inactive timeout configuration

5.3.2.15 Graceful close

The parameter `/MICROSAR/DoIP/DoIPGeneral/DoIPSupportGracefulClose` can be used to close socket connections gracefully with a TCP FIN whenever possible instead of using a TCP RST.

Note
If the ECU closes the socket connection with a FIN the TCP socket stays in TIME_WAIT state for 2x MSL until a new connection can be established.

5.3.2.16 Provide a configurable amount of user data on StartOfReception

`/MICROSAR/DoIP/DoIPGeneral/DoIPUserDataSizeOnStartOfReception` can be used to specify the size of user data that shall be forwarded to the user on `PduR_DoIPTpStartOfReception` (refer to Figure 5-38). The value must not be set to 0 Byte. By default, the value is set to 2 Byte which is in general the length of a UDS tester present.

The diagnostic application uses the forwarded data to identify tester present messages and it can therefore decide to ignore those messages instead of rejecting them completely in case it is still busy with a previous message.

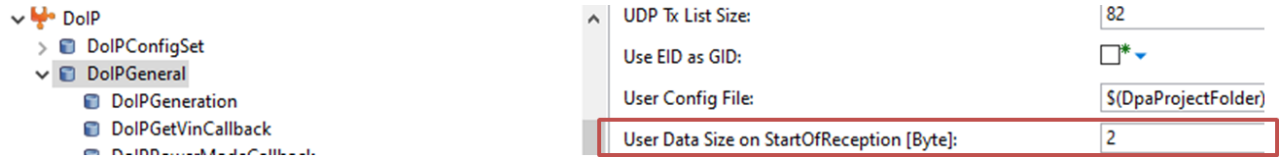

Figure 5-38 User data size on StartOfReception configuration

5.3.2.17 Optional n/k+1 Socket

This feature makes the n+1 socket for TCP and the k+1 socket for TLS optional. In this chapter and in the parameter description, the term n+1 is used for the sake of simplicity.

To make the n+1 socket optional enables optimization of resources (e.g. less socket buffers are required). The parameter `/MICROSAR/DoIP/DoIPConfigSet/DoIPInterface/DoIPConnections/DoIPTcpConnection/DoIPUsePlusOneSocket` indicates if the associated `SoAdSocketConnectionGroup` uses an n+1 socket.

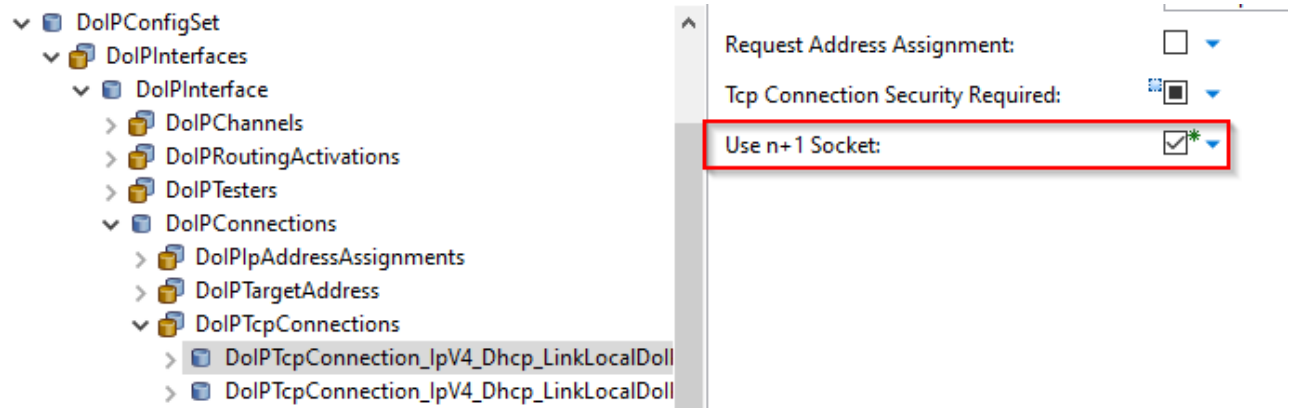

Figure 5-39 Use n+1 socket configuration

If enabled, the +1 socket is used to decline additional tester connections.

If disabled, the +1 handling is not used and all sockets can be used as tester connections.

Note
For the same `SoAdSocketConnectionGroup` this parameter must be set consistent for all related `DoIPTcpConnections`.

A missing n+1 socket increases the risk of outdated/unused tester connections which are not detected, and which will not be “cleaned up” within a proper time. Example: A tester has been connected successfully. Afterwards, the tester is removed from the network without closing the TCP connection before. The same tester is added to the network again. If just one socket is configured, the tester cannot connect again since the old connection is still active until the old connection is closed because of `T_TCP_General_Inactivity` which is specified by [5] to 5 minutes.

Caution
A missing n+1 socket increases the risk of outdated tester connections which are not detected.

There is no risk in case of the compatibility use case with a TCP and a TLS socket and if it is ensured that only one tester is in the network. In this use case the TCP socket is only used to tell the tester that TLS shall be used. The tester always connects via TCP first and its routing activation request will always be rejected since it requires a secure TLS connection (`DoIPRoutingActivationSecurityRequired`).

On reception of the routing activation request on the TCP socket the DoIP module performs an alive check on the TLS connection if the tester is already connected. This is done before the routing activation request is rejected because of the required secure TLS connection. This ensures that an outdated TLS connection is “cleaned up” by the alive check mechanism. Since a routing activation request will be always rejected in the TCP connection, no outdated TCP connection is possible.

Please note that this may lead to a scenario where an incoming routing activation request may be rejected (because of the security requirement) and an old connection is closed, too. The tester would be disconnected completely.

5.3.2.18 Manual closing of a connection

DoIP provides the API `DoIP_CloseConnection()` (chapter 4.2.16) to manually request to close a connection. This may e.g. be required on reception of OEM specific payload types on a UDP connection to be able to request to close the connection manually (refer to chapter 5.3.2.7). Depending on the “Abort” parameter it is checked for ongoing transmissions and the request is either directly forwarded to the Socket Adaptor or stored to be handled when all ongoing transmissions are finished.

Note
UDP Vehicle Announcement connections are automatically opened after they have been closed via `DoIP_CloseConnection()` API. The vehicle announcements are sent again when the connection is reopened.

5.3.2.19 Channel Ready Notification for transmission

After rebooting of the ECU the Dcm may need to transmit outstanding messages. This is only possible if the tester set up the TCP connection and activated the routing. Since Dcm does not know when the routing activation has succeeded, DoIP needs to notify the Dcm

via `BswM_DoIP_SetChannelReady()` when the related ComM channel is ready to transmit messages.

Since not all DoIPChannels are related to Dcm (i.e. in case of GW routing), the boolean parameter `/MICROSAR/DoIP/DoIPConfigSet/DoIPInterface/DoIPChannel/DoIPChannelReadyNotification` enables the channel ready notification (see Figure 5-40).

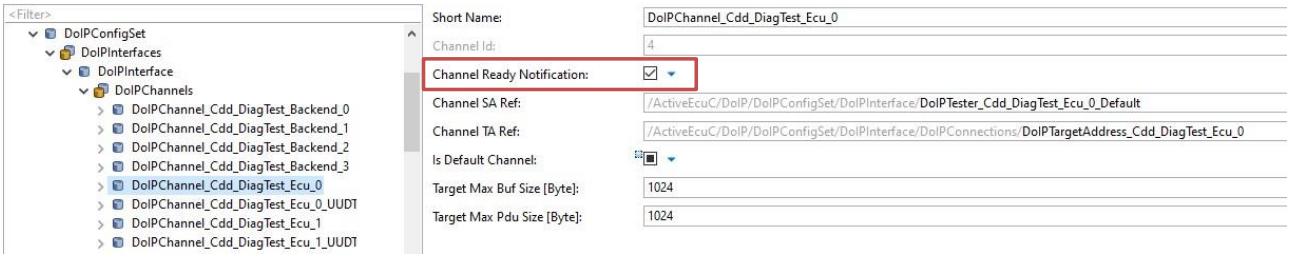

Figure 5-40 DoIP channel ready notification configuration

Moreover, the feature must be enabled in Dcm as well by configuring the parameter `/MICROSAR/Dcm/DcmConfigSet/DcmDsl/DcmDslProtocol/DcmDslProtocolRow/DcmDslConnection/DcmDslMainConnection/DcmDslChannelReadyIndicationEnabled`. For more details, please refer to the technical reference of the Dcm [9].

Additionally, it is necessary to configure a BswM rule (`BswMModeRequestPort` = “DoIP Set Channel Ready”, `BswMAction` = “Dcm Set Channel Ready”) within DaVinci Configurator (see Figure 5-41) to make sure that the notification is forwarded to the Dcm. Be aware that the “`Action List Execution`” setting must be configured to “`BSWM_CONDITION`”.

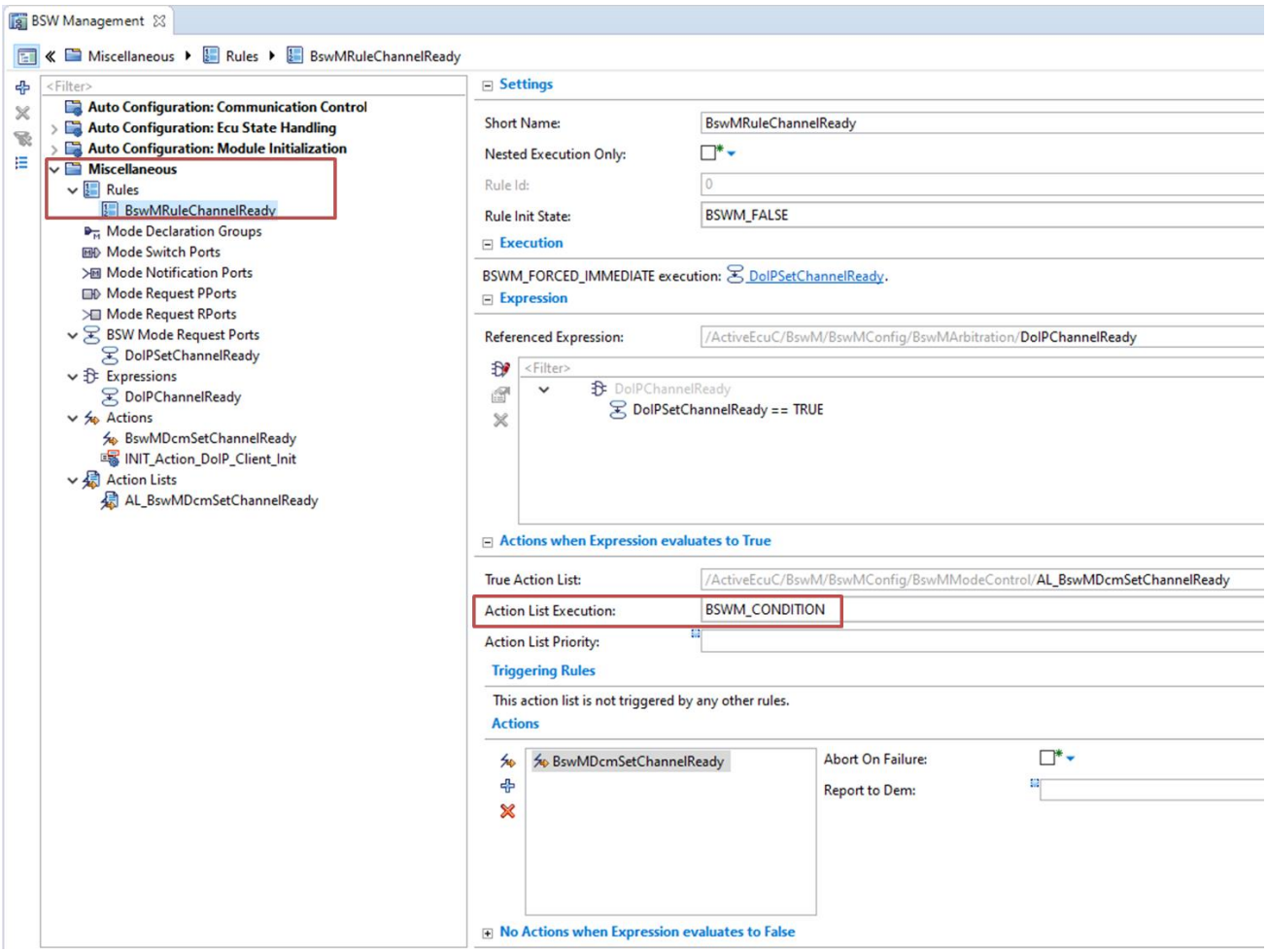

Figure 5-41 BswM rule to configure the channel ready feature

Note
DoIP calls this notification whenever a routing activation succeeds. This may result in multiple calls in case of multiple testers or if a tester reconnects again.

Note
In case the MICROSAR Dcm is not used the DoIP or BswM callout can be implemented by the user.

5.3.2.20 Configurable DoIP protocol version

The DoIP protocol version which is used on transmission inside the DoIP generic header and accepted on reception is configurable (refer to Figure 5-42). Any value from 0x02 to 0xFF is supported. Please note that 0xFF is not expected to be configured since this is the default value for the protocol version and frames with this protocol version are accepted on reception via UDP independent of the configured protocol version. Please refer to [5] and [6] for the expected value of the protocol version according to the ISO 13400-2.

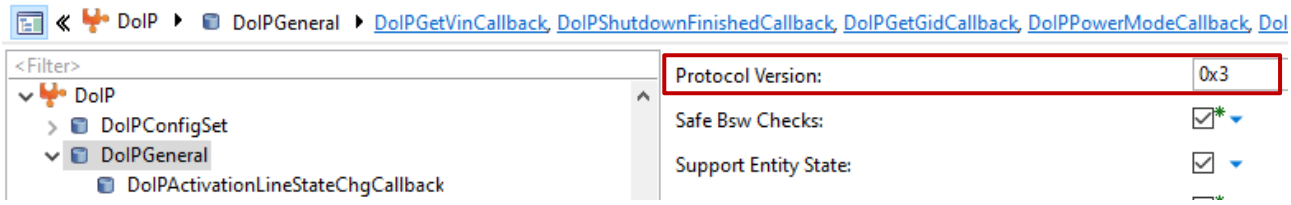

Figure 5-42 DoIP protocol version configuration

5.3.2.21 General Inactivity Time with Alive Check

The intended use of this feature is to detect unused TCP/TLS connections earlier if the n+1/k+1 socket (`/MICROSAR/DoIP/DoIPConfigSet/DoIPInterface/DoIPConnections/DoIPTcpConnection/DoIPUsePlusOneSocket`) is not used. If the feature is activated (`/MICROSAR/DoIP/DoIPConfigSet/DoIPInterface/DoIPGeneralInactivityTimeWithAliveCheck`) (refer to Figure 5-43) an alive check will be triggered before the general inactivity time (`/MICROSAR/DoIP/DoIPConfigSet/DoIPInterface/DoIPGeneralInactivityTime`) is reached. The alive check will be triggered at general inactivity time minus alive check response timeout (`/MICROSAR/DoIP/DoIPConfigSet/DoIPInterface/DoIPAliveCheckResponseTimeout`), so that if no alive check response is received after the alive check response timeout the socket will be closed after general inactivity time. If the feature is used together with a n+1/k+1 socket and an alive check is triggered by a routing activation they will not be sent out to a connection where an alive check triggered due to DoIPGeneralInactivityTimeWithAliveCheck is already pending. The routing activation is rejected after the alive check response timeout of the routing activation handler and not by an earlier alive check timeout triggered by DoIPGeneralInactivityTimeWithAliveCheck.

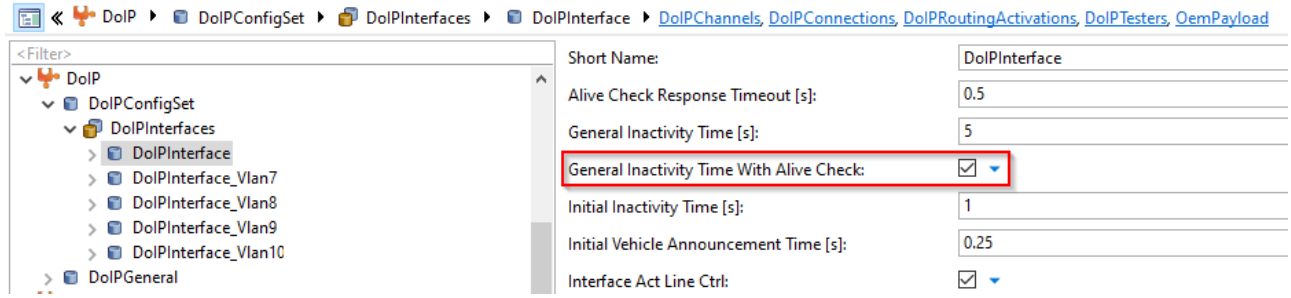

Figure 5-43 DoIP general inactivity time with alive check configuration

5.3.2.22 Use Secure Communication Callback

This callback is enabled by creating the `/MICROSAR/DoIP/DoIPGeneral/DoIpUseSecureCommunicationCallback` container. The callback is called during processing of a routing activation request if they are received over an unsecured connection but the requested routing activation requires security. Depending on the return value of the callback the user/application can decide if only secure communication is allowed (return value TRUE) or if also a routing activation from an unsecure communication channel is accepted (return value FALSE).

Figure 5-44 Use secure communication callback configuration

6 Glossary and Abbreviations

6.1 Glossary

Term	Description
DaVinci Configurator	Generation tool for MICROSAR components

Table 6-1 Glossary

6.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
BswM	Basic Software module BSW Mode Manager
DCM	Diagnostic Communication Manager
DEM	Diagnostic Event Manager
DET	Development Error Tracer
DHCP	Dynamic Host Configuration Protocol
EAD	Embedded Architecture Designer
ECU	Electronic Control Unit
HIS	Hersteller Initiative Software
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
OEM	Original Equipment Manufacturer
OS	Operating System
PDU	Protocol Data Unit
PduR	PDU Router
RTE	Runtime Environment
Rx	Reception
SoAd	Socket Adaptor
SWS	Software Specification
TCP	Transmission Control Protocol
TcpIp	TcpIp (AUTOSAR module)
Tx	Transmission
TP	Transport Protocol (AUTOSAR API)
UDP	User Datagram Protocol
UDS	Unified Diagnostic Services
VIN	Vehicle Identification Number

Table 6-2 Abbreviations

7 Contact

Visit our website for more information on

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

www.vector.com