Diagnostic over IP

Technical Reference

Version 4.4.0

Authors	Michael Dangelmaier, Jens Bauer
Status	Released

Document Information

History

Author	Date	Version	Remarks
Michael Dangelmaier	2014-05-26	1.0	Creation of document
Michael Dangelmaier	2015-08-12	1.1	Update for component release
Michael Dangelmaier	2016-01-11	2.0	Limited support of ASR4.2.2
Jens Bauer, Michael Dangelmaier	2016-02-25	2.1	Update for component release, UUDT Support according to ASR 4.2.2
Michael Dangelmaier	2016-05-19	2.2	Improved Vehicle Announcement Handling
Michael Dangelmaier	2016-09-05	3.0	Support of OEM specific payload types, Target Address Masking and Verification, Optimized TP transmission
Michael Dangelmaier	2017-01-30	3.1	Support multiple testers on different VLANs
Michael Dangelmaier	2017-05-08	4.0	Updated component history
Michael Dangelmaier	2017-05-22	4.1	Updated component history
Michael Dangelmaier	2017-07-04	4.2	PDU reception verification
Jens Bauer	2017-07-20	4.3	API Pattern, Cleanup
Jens Bauer	2017-08-21	4.4	Updated component history

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_SWS_DiagnosticOverIP.pdf	V4.2.2
[2]	AUTOSAR	AUTOSAR_SWS_DET.pdf	V3.4.1
[3]	AUTOSAR	AUTOSAR_SWS_DEM.pdf	V5.2.0
[4]	AUTOSAR	AUTOSAR_TR_BSWModuleList.pdf	V1.7.0
[5]	ISO	ISO 13400-2	IS 2012

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 Component History

The component history gives an overview over the important milestones that are supported in the different versions of the component.

Component Version	New Features
1.00	Created
1.01	Version Check
1.02	Limited support ASR4.1.3 specification
1.03	Support of Vector DoIPGw module
1.04	DoIP extracted fully from SoAd, UDP transmission queue, Shutdown mechanism
1.05	Adaptions to support ASR4.2.1 SoAd
1.06	Adaptions to support BSD socket API
2.00	Limited support of ASR4.2.2
2.01	UUDT Support according to ASR 4.2.2
2.02	Improved Vehicle Announcement Handling AUTOSAR [Issue 67021]
3.00	Support of OEM specific payload types, Target Address Masking and Verification, Optimized TP transmission
3.01	Support multiple testers on different VLANs
4.00	Reworked header includes (P3 CAD)
4.01	Support more than 255 DoIP Target Addresses
4.02	PDU reception verification (Callout for Diagnostic Firewall Use Case)
4.03	API Pattern, Cleanup
4.04	Code Refactoring

Table 1-1 Component history

2 Introduction

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

Supported AUTOSAR Release*:	4.2.2	
Supported Configuration Variants:	pre-compile	
Vendor ID:	DoIP_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	DoIP_MODULE_ID	173 decimal (according to ref. [4])

* For the detailed functional specification please also refer to the corresponding AUTOSAR SWS.

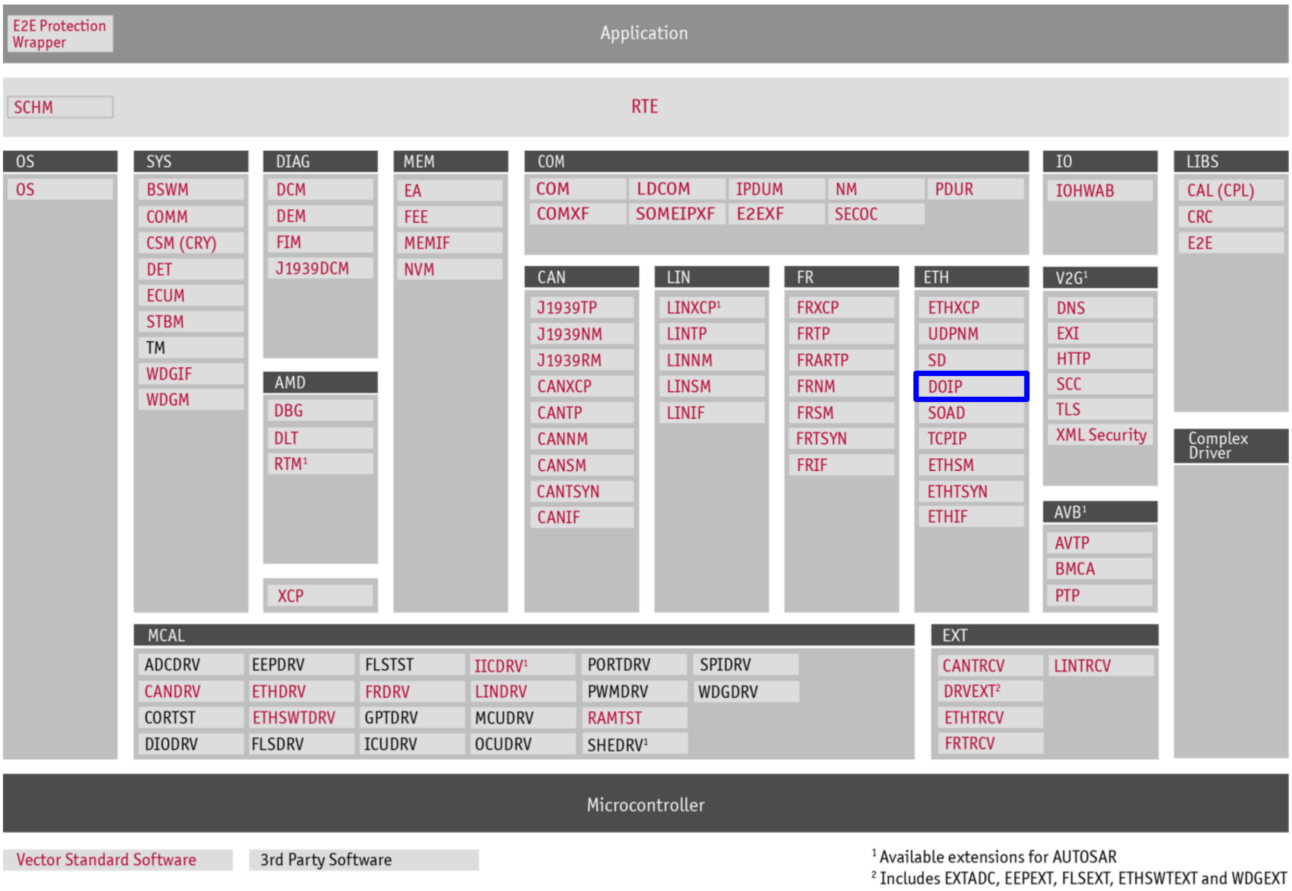
The DoIP module is a protocol to transport diagnostic data over Ethernet from tester to ECU and vice versa. It is also described in ISO Standard 13400.

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

2.1 Architecture Overview

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



[1] Available extensions for AUTOSAR

[2] Includes EXTADC, EEPEXT, FLSEXT, ETHSWTEXT and WDGEXT

Figure 2-1 AUTOSAR 4.2 Architecture Overview

Applications do not access the services of the BSW modules directly. They use the service ports provided by the BSW modules via the RTE. The service ports provided by DoIP according to AUTOSAR are currently not supported but will be available soon.

The next figure shows the interfaces to adjacent modules of the DoIP. These interfaces are described in chapter 5.

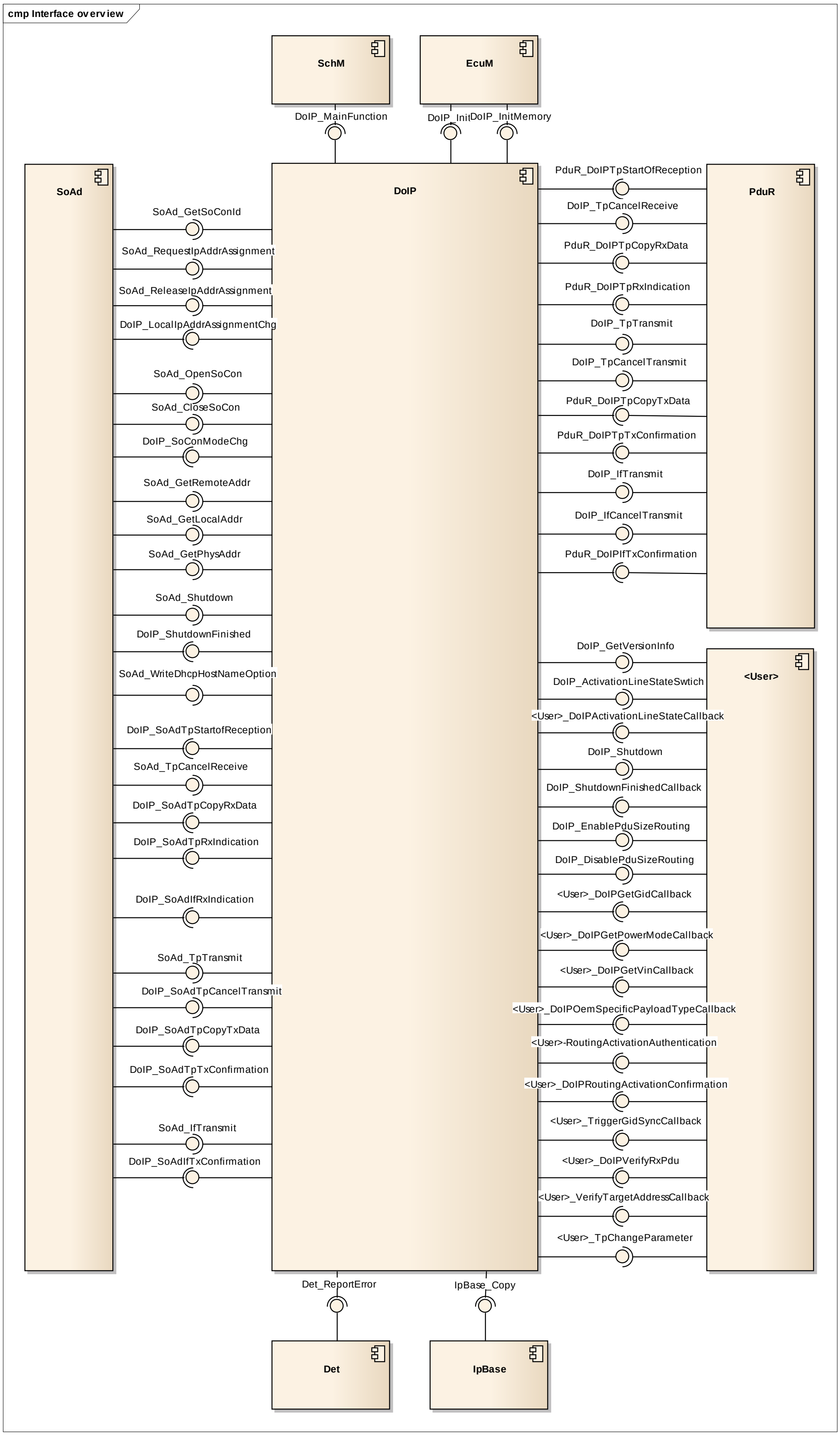

cmp Interface overview
SchM
EcuM
DoIP_MainFunction
DoIP_Init
DoIP_InitMemory
SoAd
DoIP
PduR
SoAd_GetSoConId
SoAd_RequestIpAddrAssignment
SoAd_ReleaseIpAddrAssignment
DoIP_LocalIpAddrAssignmentChg
SoAd_OpenSoCon
SoAd_CloseSoCon
DoIP_SoConModeChg
SoAd_GetRemoteAddr
SoAd_GetLocalAddr
SoAd_GetPhysAddr
SoAd_Shutdown
DoIP_ShutdownFinished
SoAd_WriteDhcpHostNameOption
DoIP_SoAdTpStartofReception
SoAd_TpCancelReceive
DoIP_SoAdTpCopyRxData
DoIP_SoAdTpRxIndication
DoIP_SoAdIfRxIndication
SoAd_TpTransmit
DoIP_SoAdTpCancelTransmit
DoIP_SoAdTpCopyTxData
DoIP_SoAdTpTxConfirmation
SoAd_IfTransmit
DoIP_SoAdIfTxConfirmation
PduR_DoIPTpStartOfReception
DoIP_TpCancelReceive
PduR_DoIPTpCopyRxData
PduR_DoIPTpRxIndication
DoIP_TpTransmit
DoIP_TpCancelTransmit
PduR_DoIPTpCopyTxData
PduR_DoIPTpTxConfirmation
DoIP_IfTransmit
DoIP_IfCancelTransmit
PduR_DoIPIfTxConfirmation
<User>
DoIP_GetVersionInfo
DoIP_ActivationLineStateSwtich
<User>_DoIPActivationLineStateCallback
DoIP_Shutdown
DoIP_ShutdownFinishedCallback
DoIP_EnablePduSizeRouting
DoIP_DisablePduSizeRouting
<User>_DoIPGetGidCallback
<User>_DoIPGetPowerModeCallback
<User>_DoIPGetVinCallback
<User>_DoIPOemSpecificPayloadTypeCallback
<User>-RoutingActivationAuthentication
<User>_DoIPRoutingActivationConfirmation
<User>_TriggerGidSyncCallback
<User>_DoIPVerifyRxPdu
<User>_VerifyTargetAddressCallback
<User>_TpChangeParameter
Det_ReportError
IpBase_Copy
Det
IpBase

Figure 2-2 Interfaces to adjacent modules of the DoIP

3 Functional Description

3.1 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 3-1 Supported AUTOSAR standard conform features
- Table 3-2 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 3-3 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
DoIP Activation Line with IPv4
Routing Activation handling for OEM specific part
UUDT over IF-API

Table 3-1 Supported AUTOSAR standard conform features

3.1.1 Deviations

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

Not Supported AUTOSAR Standard Conform Features
DoIP Service Component
DoIP Activation Line with IPv6

Table 3-2 Not supported AUTOSAR standard conform features

3.1.2 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
ISO Version support for DIS, FDIS and IS

Features Provided Beyond The AUTOSAR Standard
Default Tester
IPv4/6 and VLAN support
Shutdown mechanism
OEM specific payload types
Target Address Masking and Verification
Optimized TP transmission
PDU reception verification

Table 3-3 Features provided beyond the AUTOSAR standard

3.1.3 Known Issues (Low Priority)

There are no known issues.

3.1.4 Hints

3.1.4.1 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 5 for details.

3.1.4.2 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.

3.2 Initialization

DoIP is initialized via a call of `DoIP_InitMemory()` followed by the call of `DoIP_Init()`. The current version does not support link-time or post-build configuration so `DoIP_Init()` can be called with a `NULL_PTR`.

3.3 States

The DoIP has no specific state handling after calling the initialization functions described in chapter 3.2.

3.4 Main Functions

Within the main function 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

3.5 Error Handling

3.5.1 Development Error Reporting

By default, development errors are reported to the DET using the service `Det_ReportError()` as specified in [2], if development error reporting is enabled (i.e. pre-compile parameter `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 5.2. The following table presents the service IDs and the related services:

Service ID	Service
0x00	DOIP_SID_GET_VERSION_INFO
0x01	DOIP_SID_INIT
0x02	DOIP_SID_MAIN_FUNCTION
0x03	DOIP_SID_TP_TRANSMIT
0x04	DOIP_SID_TP_CANCEL_TRANSMIT
0x05	DOIP_SID_TP_CANCEL_RECEIVE
0x0B	DOIP_SID_SO_CON_MODE_CHG
0x0C	DOIP_SID_LOC_IP_ADDR_ASSIGN_CHG
0x0F	DOIP_SID_ACTIVATION_LINE_SWITCH
0x40	DOIP_SID_IF_TX_CONFIRMATION
0x42	DOIP_SID_IF_RX_INDICATION
0x43	DOIP_SID_TP_COPY_TX_DATA
0x44	DOIP_SID_TP_COPY_RX_DATA
0x45	DOIP_SID_TP_RX_INDICATION
0x46	DOIP_SID_TP_START_OF_RECEPTION
0x48	DOIP_SID_TP_TX_CONFIRMATION

Service ID	Service
0x49	DOIP_SID_IF_TRANSMIT
0x4A	DOIP_SID_IF_CANCEL_TRANSMIT
0xEB	DOIP_SID_VACTIVATION_LINE_TO_ACTIVE
0xEC	DOIP_SID_ENABLE_PDU_SIZE_ROUTING
0xED	DOIP_SID_DISABLE_PDU_SIZE_ROUTING
0xEE	DOIP_SID_VUDP_SINGLE_TRANSMIT
0xEF	DOIP_SID_VTCP_TRANSMIT

Table 3-4 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

Table 3-5 Errors reported to DET

3.5.2 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. Each production error can be enabled and disabled separately.

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.

4 Integration

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

4.1 Scope of Delivery

The delivery of DoIP contains the files which are described in the chapters 4.1.1 and 4.1.2:

4.1.1 Static Files

File Name	Description
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
_Appl_DoIP.h	Template header file for callout functions

Table 4-1 Static files

4.1.2 Dynamic Files

The dynamic files are generated by the configuration tool Configurator 5.

File Name	Description
DoIP_Cfg.h	Header file for configuration parameter and defines (e.g. feature switches)
DoIP_Lcfg.h	Header file for generated source code
DoIP_Lcfg.c	Source file for generated source code

Table 4-2 Generated files

4.2 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 medium.

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.

5 API Description

For an interfaces overview please see Figure 2-2.

5.1 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_ActivationLineType	uint8	Activation line state	0u DOIP_ACTIVATION_LINE_INACTIVE
			1u DOIP_ACTIVATION_LINE_ACTIVE
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

Table 5-1 Type definitions

5.2 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. Services used by DoIP

5.2.1 DoIP_InitMemory

Prototype	
`void DoIP_InitMemory (void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
Power-up memory initialization.	

Particularities and Limitations
Use this function in case these variables are not initialized by the startup code. Module is uninitialized. Initializes component variables in *_INIT_* sections at power up.
Call context
> TASK > This function is Synchronous > This function is Non-Reentrant

Table 5-2 DoIP_InitMemory

5.2.2 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.	
Particularities and Limitations	
> Interrupts are disabled.Module is uninitialized.DoIP_InitMemory has been called unless DoIP_State is initialized by start-up code. 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.	
Call context	
> TASK > This function is Synchronous > This function is Non-Reentrant	

Table 5-3 DoIP_Init

5.2.3 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.
Particularities and Limitations
- Returns version information, vendor ID and AUTOSAR module ID of the component. Configuration Variant(s): DOIP_VERSION_INFO_API
Call context
> TASK|ISR2 > This function is Synchronous > This function is Reentrant

Table 5-4 DoIP_GetVersionInfo

5.2.4 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.	
Particularities and Limitations	
- 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.	
Call context	
> TASK > This function is Non-Reentrant	

Table 5-5 DoIP_TpTransmit

5.2.5 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.	
Particularities and Limitations	
- 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.	
Call context	
> TASK > This function is Non-Reentrant	

Table 5-6 DoIP_TpCancelTransmit

5.2.6 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.	
Particularities and Limitations	
- 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.	
Call context	
> TASK > This function is Non-Reentrant	

Table 5-7 DoIP_TpCancelReceive

5.2.7 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	

Table 5-8 DoIP_IfTransmit

5.2.8 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 Non-Reentrant	

Table 5-9 DoIP_IfCancelTransmit

5.2.9 DoIP_TpChangeParameter

Prototype	
Std_ReturnType **DoIP_TpChangeParameter** (PduIdType id, TPParameterType parameter, uint16 value)	
Parameter	
id [in]	Identification of the TP PDU.
parameter [in]	Parameter identifier
value [in]	Parameter value
Return code	
Std_ReturnType	E_OK Request is accepted.
Std_ReturnType	E_NOT_OK Request is not accepted.
Functional Description	
Implemented to support generic PduR modules.	
Particularities and Limitations	
No functionality is implemented. - -	
Call context	
> TASK > This function is Synchronous > This function is Reentrant	

Table 5-10 DoIP_TpChangeParameter

5.2.10 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_VSUPPORT_PDU_SIZE_ROUTING	
Call context	
> TASK	

> This function is Synchronous > This function is Reentrant

Table 5-11 DoIP_EnablePduSizeRouting

5.2.11 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_VSUPPORT_PDU_SIZE_ROUTING	
Call context	
> TASK > This function is Synchronous > This function is Reentrant	

Table 5-12 DoIP_DisablePduSizeRouting

5.2.12 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.	
Particularities and Limitations	
- All sockets will be closed and modules change to special shutdown state. Configuration Variant(s): DOIP_VSUPPORT_SHUTDOWN	

Call context
> TASK > This function is Non-Reentrant

Table 5-13 DoIP_Shutdown

5.2.13 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 5-14 DoIP_MainFunction

5.3 Services provided 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
IpBase	IpBaseCopy
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

Component	API
SoAd	SoAd_CloseSoCon
SoAd	SoAd_GetLocalAddr
SoAd	SoAd_GetPhysAddr
SoAd	SoAd_GetRemoteAddr
SoAd	SoAd_GetSoConId
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

Table 5-15 Services used by the DoIP

5.4 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.

5.4.1 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.	

<table>
<tr><th>Particularities and Limitations</th></tr>
<tr><td>-
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.</td></tr>
<tr><td>Call context</td></tr>
<tr><td>&gt; TASK|ISR2
> This function is Synchronous
> This function is Reentrant</td></tr>
</table>

Table 5-16 DoIP_SoAdTpCopyTxData

5.4.2 DoIP_SoAdTpTxConfirmation

<table>
<tr><th colspan="2">Prototype</th></tr>
<tr><td colspan="2"><code>void <b>DoIP_SoAdTpTxConfirmation</b> (PduIdType id, Std_ReturnType result)</code></td></tr>
<tr><th colspan="2">Parameter</th></tr>
<tr><td>id [in]</td><td>Identification of the transmitted TP PDU.</td></tr>
<tr><td>result [in]</td><td>Result of the transmission of the TP PDU.</td></tr>
<tr><th colspan="2">Return code</th></tr>
<tr><td>void</td><td>none</td></tr>
<tr><th colspan="2">Functional Description</th></tr>
<tr><td colspan="2">Transmission confirmation callback for TP.</td></tr>
<tr><th colspan="2">Particularities and Limitations</th></tr>
<tr><td colspan="2">-
This function is called after the TP PDU has been transmitted on its network, the result indicates whether the transmission was successful or not.</td></tr>
<tr><td colspan="2">Call context</td></tr>
<tr><td colspan="2">&gt; TASK|ISR2
> This function is Synchronous
> This function is Reentrant</td></tr>
</table>

Table 5-17 DoIP_SoAdTpTxConfirmation

5.4.3 DoIP_SoAdTpCopyRxData

<table>
<tr><th colspan="2">Prototype</th></tr>
<tr><td colspan="2"><code>BufReq_ReturnType <b>DoIP_SoAdTpCopyRxData</b> (PduIdType id, PduInfoType *info, PduLengthType *bufferSizePtr)</code></td></tr>
<tr><th colspan="2">Parameter</th></tr>
<tr><td>id [in]</td><td>Identification of the received TP PDU.</td></tr>
<tr><td>info [in]</td><td>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</td></tr>
</table>

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

Table 5-18 DoIP_SoAdTpCopyRxData

5.4.4 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 by SoAd)
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.	
Particularities and Limitations	
- 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.	

Call context
> TASK|ISR2 > This function is Synchronous > This function is Reentrant

Table 5-19 DoIP_SoAdTpStartOfReception

5.4.5 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.
Return code	
void	none
Functional Description	
Receive indication callback for TP.	
Particularities and Limitations	
- Called after an TP PDU has been received via the TP API, the result indicates whether the transmission was successful or not.	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Reentrant	

Table 5-20 DoIP_SoAdTpRxIndication

5.4.6 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.	

Particularities and Limitations
- Indication of a received IF PDU from a lower layer communication interface module.
Call context
> TASK|ISR2 > This function is Synchronous > This function is Non-Reentrant

Table 5-21 DoIP_SoAdIfRxIndication

5.4.7 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.	
Particularities and Limitations	
No functionality is implemented. - The lower layer communication interface module confirms the transmission of an IF PDU.	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Reentrant	

Table 5-22 DoIP_SoAdIfTxConfirmation

5.4.8 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
Return code	
void	none

Functional Description
Socket state change callback.
Particularities and Limitations
- Notification about a SoAd socket connection state change, e.g. socket connection gets online.
Call context
> TASK|ISR2 > This function is Synchronous > This function is Non-Reentrant

Table 5-23 DoIP_SoConModeChg

5.4.9 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.
Return code	
void	none
Functional Description	
IP address assignment change callback.	
Particularities and Limitations	
- 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).	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Non-Reentrant	

Table 5-24 DoIP_LocalIpAddrAssignmentChg

5.4.10 DoIP_ActivationLineSwitch

Prototype	
`void` **`DoIP_ActivationLineSwitch`** `(void)`	
Parameter	
void	none

Return code	
void	none
Functional Description	
Notifies DoIP on a switch of the DoIPActivationLine.	
Particularities and Limitations	
- This function is used to notify the DoIP on a switch of the DoIPActivationLine.	
Call context	
> TASK|ISR2 > This function is Synchronous > This function is Non-Reentrant	

Table 5-25 DoIP_ActivationLineSwitch

5.4.11 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 5-26 DoIP_ShutdownFinished

5.5 Configurable Interfaces

5.5.1 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:

5.5.1.1 <User>_DoIPGetVinCallback

<table>
<tr><th colspan="3">Prototype</th></tr>
<tr><td colspan="3"><code>Std_ReturnType <b>[DoIPGetVinCallbackName]</b> (uint8 *Vin)</code></td></tr>
<tr><th colspan="3">Parameter</th></tr>
<tr><td>Vin [in]</td><td colspan="2">Pointer to buffer where the VIN shall be stored.</td></tr>
<tr><th colspan="3">Return code</th></tr>
<tr><td rowspan="2">Std_ReturnType</td><td>E_OK</td><td>Request is accepted.</td></tr>
<tr><td>E_NOT_OK</td><td>Request is not accepted.</td></tr>
<tr><th colspan="3">Functional Description</th></tr>
<tr><td colspan="3">Retrieves VIN from application.</td></tr>
<tr><th colspan="3">Particularities and Limitations</th></tr>
<tr><td colspan="3"></td></tr>
<tr><td colspan="3">Call context</td></tr>
<tr><td colspan="3">> TASK|ISR2
> This function is Synchronous
> This function is Non-Reentrant</td></tr>
</table>

Table 5-27 <User>_DoIPGetVinCallback

5.5.1.2 <User>_DoIPGetGidCallback

<table>
<tr><th colspan="3">Prototype</th></tr>
<tr><td colspan="3"><code>Std_ReturnType <b>[DoIPGetGidCallbackName]</b> (uint8 *GroupId)</code></td></tr>
<tr><th colspan="3">Parameter</th></tr>
<tr><td>GroupId [in]</td><td colspan="2">Pointer to buffer where the GID shall be stored.</td></tr>
<tr><th colspan="3">Return code</th></tr>
<tr><td rowspan="2">Std_ReturnType</td><td>E_OK</td><td>Request is accepted.</td></tr>
<tr><td>E_NOT_OK</td><td>Request is not accepted.</td></tr>
<tr><th colspan="3">Functional Description</th></tr>
<tr><td colspan="3">Retrieves GID from application.</td></tr>
<tr><th colspan="3">Particularities and Limitations</th></tr>
<tr><td colspan="3"></td></tr>
<tr><td colspan="3">Call context</td></tr>
<tr><td colspan="3">> TASK|ISR2
> This function is Synchronous
> This function is Non-Reentrant</td></tr>
</table>

Table 5-28 <User>_DoIPGetGidCallback

5.5.1.3 <User>_DoIPTriggerGidSyncCallback

<table>
<tr><th colspan="3">Prototype</th></tr>
<tr><td colspan="3"><code>Std_ReturnType <b>[DoIPTriggerGidSyncCallbackName]</b> (void)</code></td></tr>
<tr><th colspan="3">Parameter</th></tr>
<tr><td>void [in]</td><td colspan="2">none</td></tr>
<tr><th colspan="3">Return code</th></tr>
<tr><td rowspan="2">Std_ReturnType</td><td>E_OK</td><td>GroupIdentifier Synchronization was triggered.</td></tr>
<tr><td>E_NOT_OK</td><td>GroupIdentifier Synchronization could not be triggered so try again next MainFunction.</td></tr>
<tr><th colspan="3">Functional Description</th></tr>
<tr><td colspan="3">Triggers GID synchronization at application.</td></tr>
<tr><th colspan="3">Particularities and Limitations</th></tr>
<tr><td colspan="3"></td></tr>
<tr><th colspan="3">Call context</th></tr>
<tr><td colspan="3">> TASK|ISR2
> This function is Synchronous
> This function is Non-Reentrant</td></tr>
</table>

Table 5-29 <User>_DoIPTriggerGidSyncCallback

5.5.1.4 <User>_DoIPGetPowerModeCallback

<table>
<tr><th colspan="3">Prototype</th></tr>
<tr><td colspan="3"><code>Std_ReturnType <b>[DoIPGetPowerModeCallbackName]</b> (DoIP_PowerStateType *PowerStateReady)</code></td></tr>
<tr><th colspan="3">Parameter</th></tr>
<tr><td>PowerStateReady [in]</td><td colspan="2">Pointer to buffer where the power mode shall be stored.</td></tr>
<tr><th colspan="3">Return code</th></tr>
<tr><td rowspan="2">Std_ReturnType</td><td>E_OK</td><td>Request is accepted.</td></tr>
<tr><td>E_NOT_OK</td><td>Request is not accepted.</td></tr>
<tr><th colspan="3">Functional Description</th></tr>
<tr><td colspan="3">Retrieves power mode from application.</td></tr>
<tr><th colspan="3">Particularities and Limitations</th></tr>
<tr><td colspan="3"></td></tr>
<tr><th colspan="3">Call context</th></tr>
<tr><td colspan="3">> TASK|ISR2
> This function is Synchronous
> This function is Non-Reentrant</td></tr>
</table>

Table 5-30 <User>_DoIPGetPowerModeCallback

5.5.1.5 <User>_DoIPShutdownFinishedCallback

Prototype		
`Std_ReturnType` **`[DoIPShutdownFinishedCallbackName]`** `(void)`		
Parameter		
void [in]	none	
Return code		
Std_ReturnType	E_OK	Request is accepted.
	E_NOT_OK	Request is not accepted.
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 5-31 <User>_DoIPShutdownFinishedCallback

5.5.1.6 <User>_DoIPActivationLineStateCallback

Prototype		
`Std_ReturnType` **`[DoIPActivationLineStateCallbackName]`** `(DoIP_ActivationLineType *const state)`		
Parameter		
state [in]	Pointer to buffer where activation line state shall be stored.	
Return code		
Std_ReturnType	E_OK	Request is accepted.
	E_NOT_OK	Request is not accepted.
Functional Description		
Retrieves activation line state from application. It can be configured in configuration container `DoIPActivationLineStateCallback`.		
Particularities and Limitations		
Call context		
> TASK|ISR2 > This function is Synchronous > This function is Non-Reentrant		

Table 5-32 <User>_DoIPActivationLineStateCallback

5.5.1.7 <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.	
AuthenticationResData [in]	Pointer to OEM specific part for authentication of routing activation response.	
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 5-33 <User>_DoIPRoutingActivationAuthentication

5.5.1.8 <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.	
ConfirmationResData [in]	Pointer to OEM specific part for authentication of routing activation response.	
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 5-34 <User>_DoIPRoutingActivationConfirmation

5.5.1.9 <User>_DoIPOemSpecificPayloadTypeCallback

Prototype		
`Std_ReturnType` **`[DoIPOemSpecificPayloadTypeCallbackName]`** `(uint16 RxPayloadType, PduInfoType* RxUserData, DoIP_OemPayloadTypeFlagType Flags, uint16* TxPayloadType, PduInfoType* TxUserData)`		
Parameter		
RxPayloadType [in]	Received payload type.	
RxUserData [in]	Pointer to received user data.	
Flags [in]	Flags indicates protocol (TCP/UDP) and routing activation state.	
TxPayloadType [out]	Payload type for response which must not to be set if no response shall be sent.	
TxUserData [in/out]	Pointer to buffer where user can store user data for response. As “in” parameter it indicates the buffer size provided by DoIP and must be set to length of copied response data.	
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 and initiate transmission of a response.		
Particularities and Limitations		
Call context		
> TASK|ISR2 > This function is Synchronous > This function is Non-Reentrant		

Table 5-35 <User>_DoIPOemSpecificPayloadTypesCallback

5.5.1.10 <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 5-36 <User>_DoIPVerifyTargetAddressCallback

5.5.1.11 <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 5-37 <User>_DoIPVerifyRxPduCallback

6 Configuration

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

- Configurator 5

6.1 Configuration Variants

The DoIP supports the configuration variants

- `VARIANT-PRE-COMPILE`

6.2 Configuration Procedure

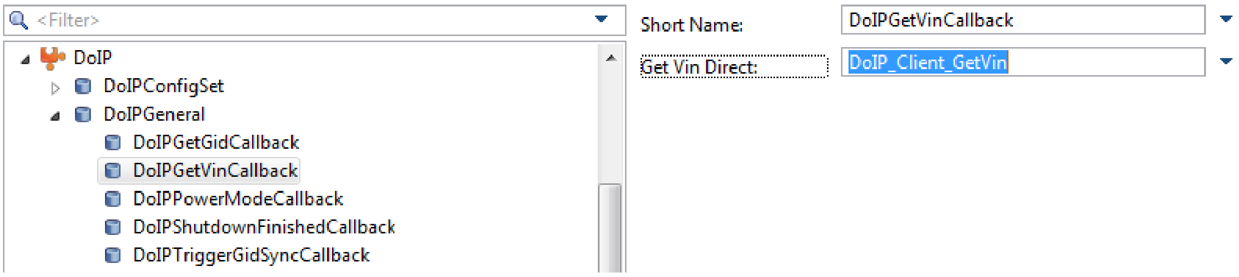
This chapter gives a short introduction of how to configure DoIP.

6.2.1 Basic functionality

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

6.2.1.1 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.



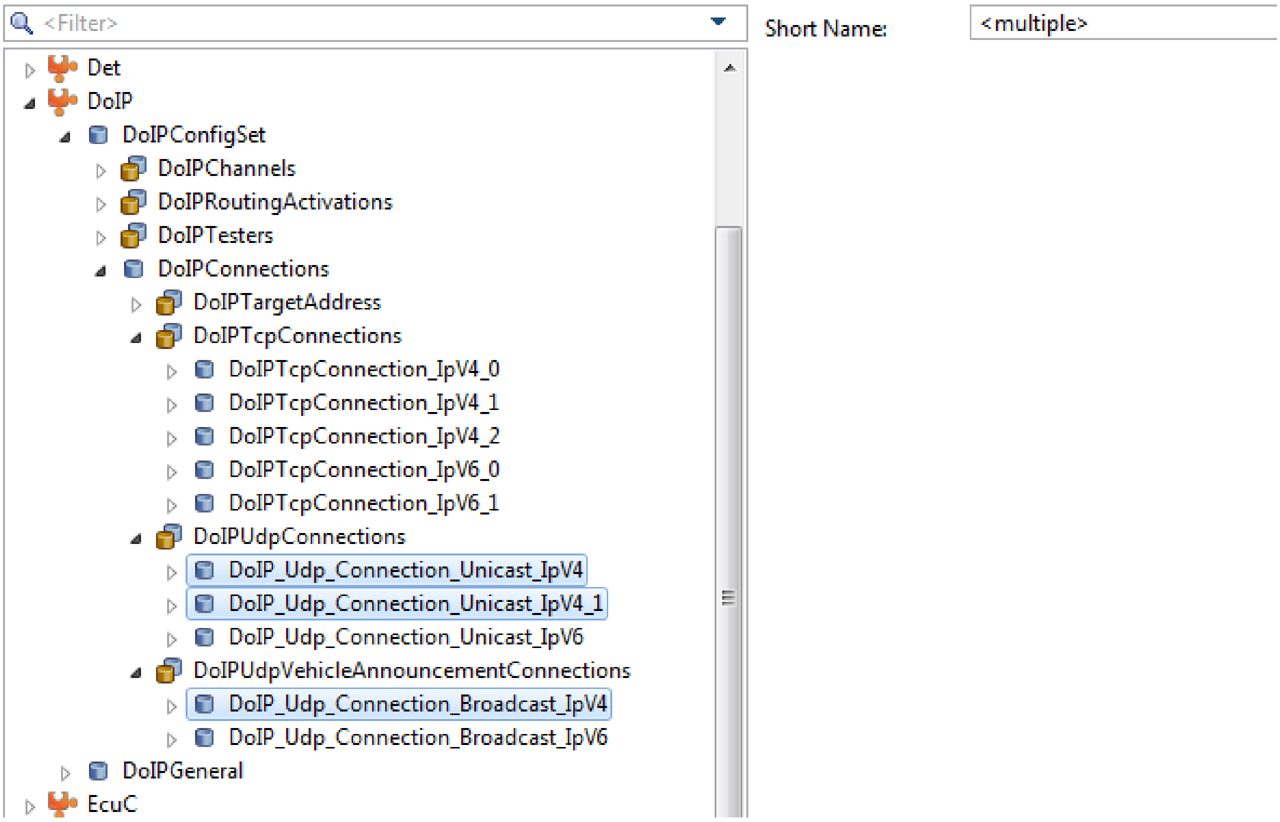
Short Name: DoIPGetVinCallback

Get Vin Direct: DoIP_Client_GetVin

Figure 6-1 Configure callback functions

6.2.1.2 TCP/UDP connection configuration (interface to SoAd)

First it is required to configure exactly one UDP Vehicle Announcement Connection to handle Vehicle Announcement messages and at least 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 6-2.



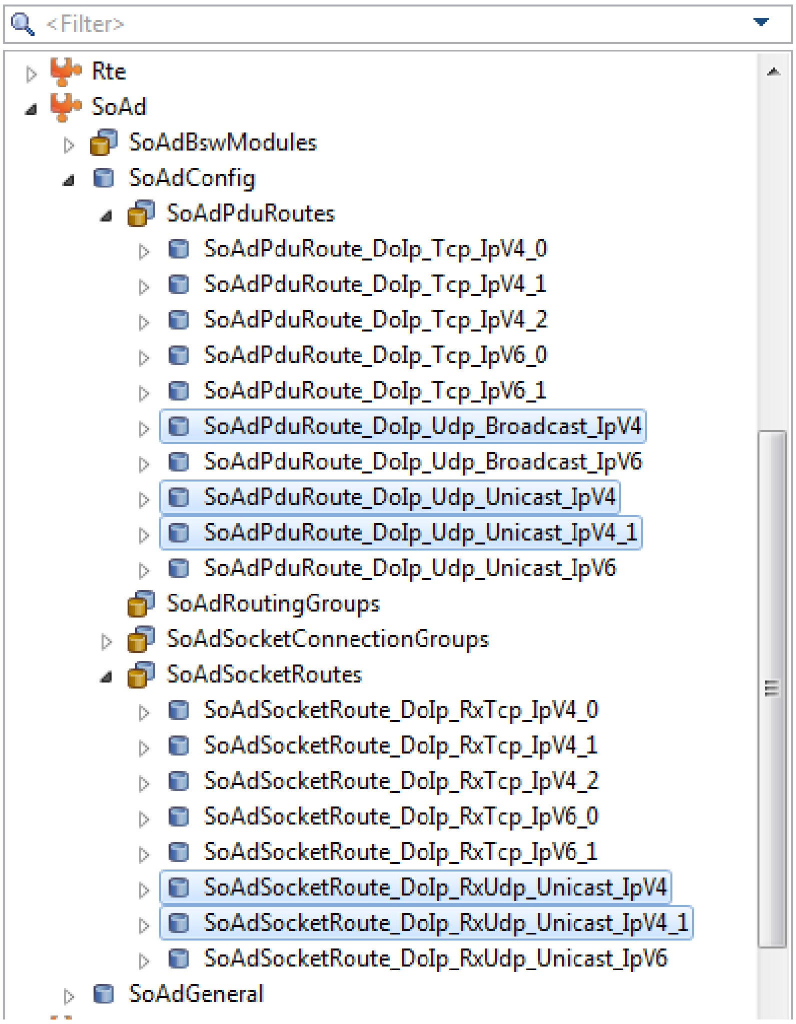
Short Name: <multiple>

- Det
- DoIP
  - DoIPConfigSet
    - DoIPChannels
    - DoIPRoutingActivations
    - DoIPTesters
    - DoIPConnections
      - DoIPTargetAddress
      - DoIPTcpConnections
        - DoIPTcpConnection_IpV4_0
        - DoIPTcpConnection_IpV4_1
        - DoIPTcpConnection_IpV4_2
        - DoIPTcpConnection_IpV6_0
        - DoIPTcpConnection_IpV6_1
      - DoIPUdpConnections
        - DoIP_Udp_Connection_Unicast_IpV4
        - DoIP_Udp_Connection_Unicast_IpV4_1
        - DoIP_Udp_Connection_Unicast_IpV6
      - DoIPUdpVehicleAnnouncementConnections
        - DoIP_Udp_Connection_Broadcast_IpV4
        - DoIP_Udp_Connection_Broadcast_IpV6
  - DoIPGeneral
- EcuC

Figure 6-2 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 6-3).


<Filter>
Rte
SoAd
SoAdBswModules
SoAdConfig
SoAdPduRoutes
SoAdPduRoute_DoIp_Tcp_IpV4_0
SoAdPduRoute_DoIp_Tcp_IpV4_1
SoAdPduRoute_DoIp_Tcp_IpV4_2
SoAdPduRoute_DoIp_Tcp_IpV6_0
SoAdPduRoute_DoIp_Tcp_IpV6_1
SoAdPduRoute_DoIp_Udp_Broadcast_IpV4
SoAdPduRoute_DoIp_Udp_Broadcast_IpV6
SoAdPduRoute_DoIp_Udp_Unicast_IpV4
SoAdPduRoute_DoIp_Udp_Unicast_IpV4_1
SoAdPduRoute_DoIp_Udp_Unicast_IpV6
SoAdRoutingGroups
SoAdSocketConnectionGroups
SoAdSocketRoutes
SoAdSocketRoute_DoIp_RxTcp_IpV4_0
SoAdSocketRoute_DoIp_RxTcp_IpV4_1
SoAdSocketRoute_DoIp_RxTcp_IpV4_2
SoAdSocketRoute_DoIp_RxTcp_IpV6_0
SoAdSocketRoute_DoIp_RxTcp_IpV6_1
SoAdSocketRoute_DoIp_RxUdp_Unicast_IpV4
SoAdSocketRoute_DoIp_RxUdp_Unicast_IpV4_1
SoAdSocketRoute_DoIp_RxUdp_Unicast_IpV6
SoAdGeneral

Short Name:

PDU:

PDU Header ID:

PDU ID:

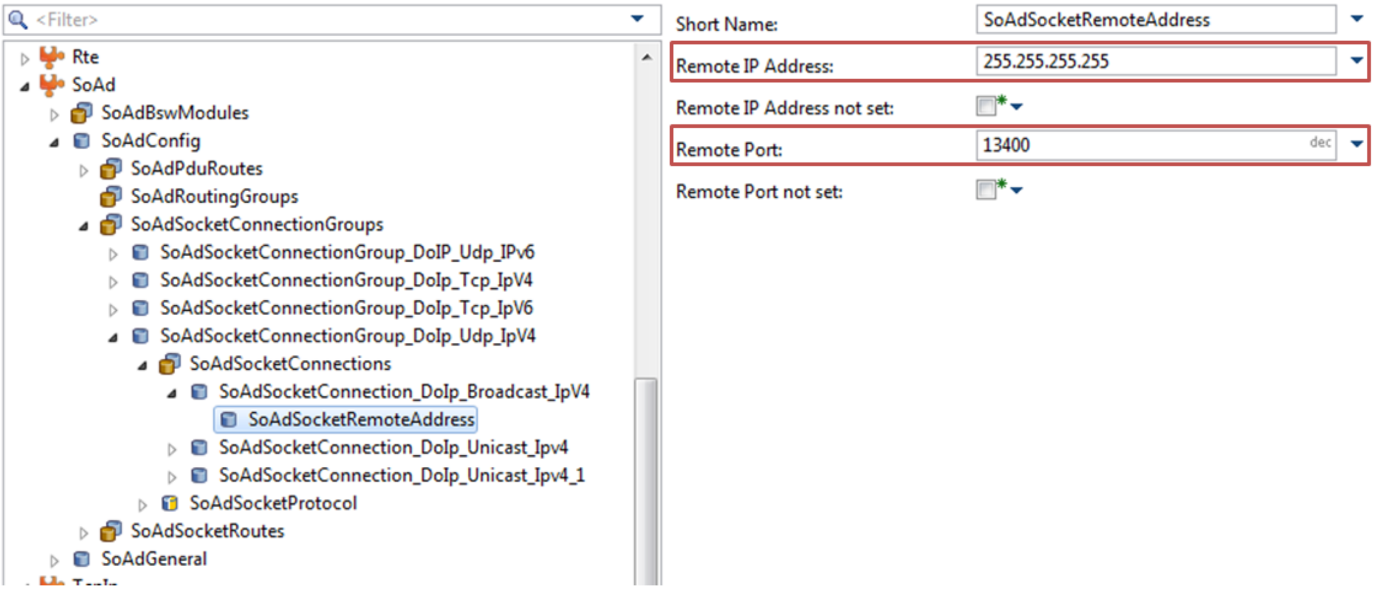
Rx Socket Conn Or Socket Conn Bundle Ref

Upper Layer Type:

Figure 6-3 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 6-4).



```
<Filter>
▷ Rte
◢ SoAd
    ▷ SoAdBswModules
    ◢ SoAdConfig
        ▷ SoAdPduRoutes
          SoAdRoutingGroups
        ◢ SoAdSocketConnectionGroups
            ▷ SoAdSocketConnectionGroup_DoIP_Udp_IPv6
            ▷ SoAdSocketConnectionGroup_DoIp_Tcp_IpV4
            ▷ SoAdSocketConnectionGroup_DoIp_Tcp_IpV6
            ◢ SoAdSocketConnectionGroup_DoIp_Udp_IpV4
                ◢ SoAdSocketConnections
                    ◢ SoAdSocketConnection_DoIp_Broadcast_IpV4
                          SoAdSocketRemoteAddress
                    ▷ SoAdSocketConnection_DoIp_Unicast_Ipv4
                    ▷ SoAdSocketConnection_DoIp_Unicast_Ipv4_1
                ▷ SoAdSocketProtocol
        ▷ SoAdSocketRoutes
    ▷ SoAdGeneral
```

Short Name: SoAdSocketRemoteAddress

Remote IP Address: 255.255.255.255

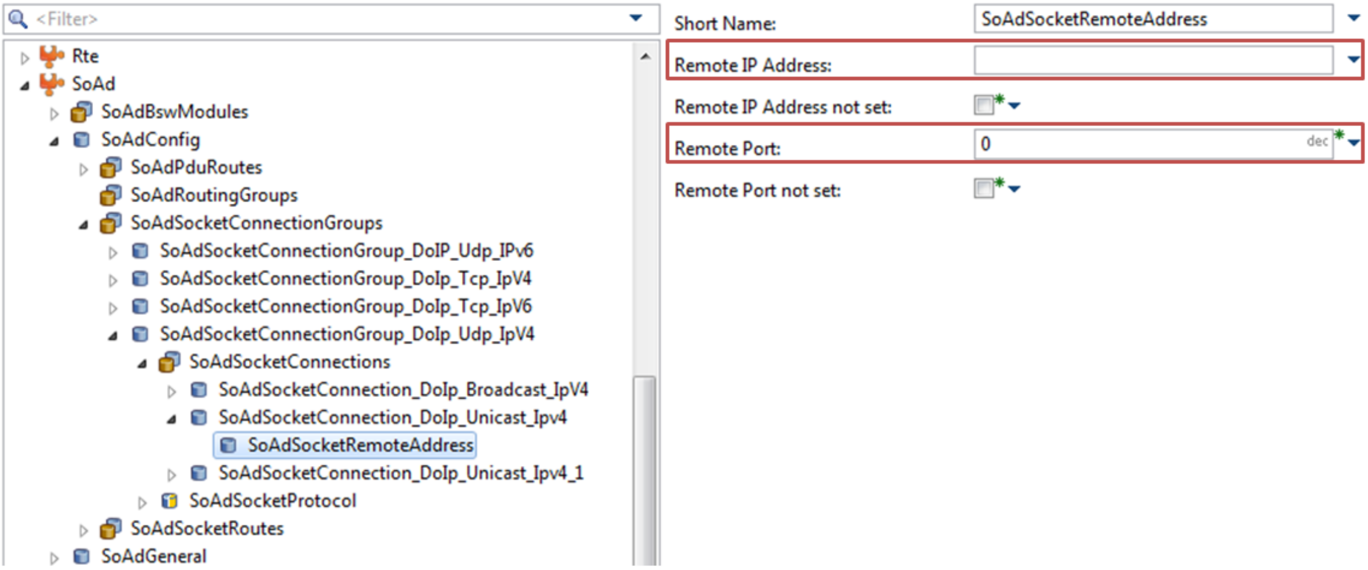
Remote IP Address not set: □

Remote Port: 13400 dec

Remote Port not set: □

Figure 6-4 UDP Vehicle Announcement connection remote address configuration

UDP connections have to configure a remote address set to wildcard (Figure 6-5).



```
<Filter>
▷ Rte
◢ SoAd
    ▷ SoAdBswModules
    ◢ SoAdConfig
        ▷ SoAdPduRoutes
          SoAdRoutingGroups
        ◢ SoAdSocketConnectionGroups
            ▷ SoAdSocketConnectionGroup_DoIP_Udp_IPv6
            ▷ SoAdSocketConnectionGroup_DoIp_Tcp_IpV4
            ▷ SoAdSocketConnectionGroup_DoIp_Tcp_IpV6
            ◢ SoAdSocketConnectionGroup_DoIp_Udp_IpV4
                ◢ SoAdSocketConnections
                    ▷ SoAdSocketConnection_DoIp_Broadcast_IpV4
                    ◢ SoAdSocketConnection_DoIp_Unicast_Ipv4
                          SoAdSocketRemoteAddress
                    ▷ SoAdSocketConnection_DoIp_Unicast_Ipv4_1
                ▷ SoAdSocketProtocol
        ▷ SoAdSocketRoutes
    ▷ SoAdGeneral
```

Short Name: SoAdSocketRemoteAddress

Remote IP Address:

Remote IP Address not set: □

Remote Port: 0 dec

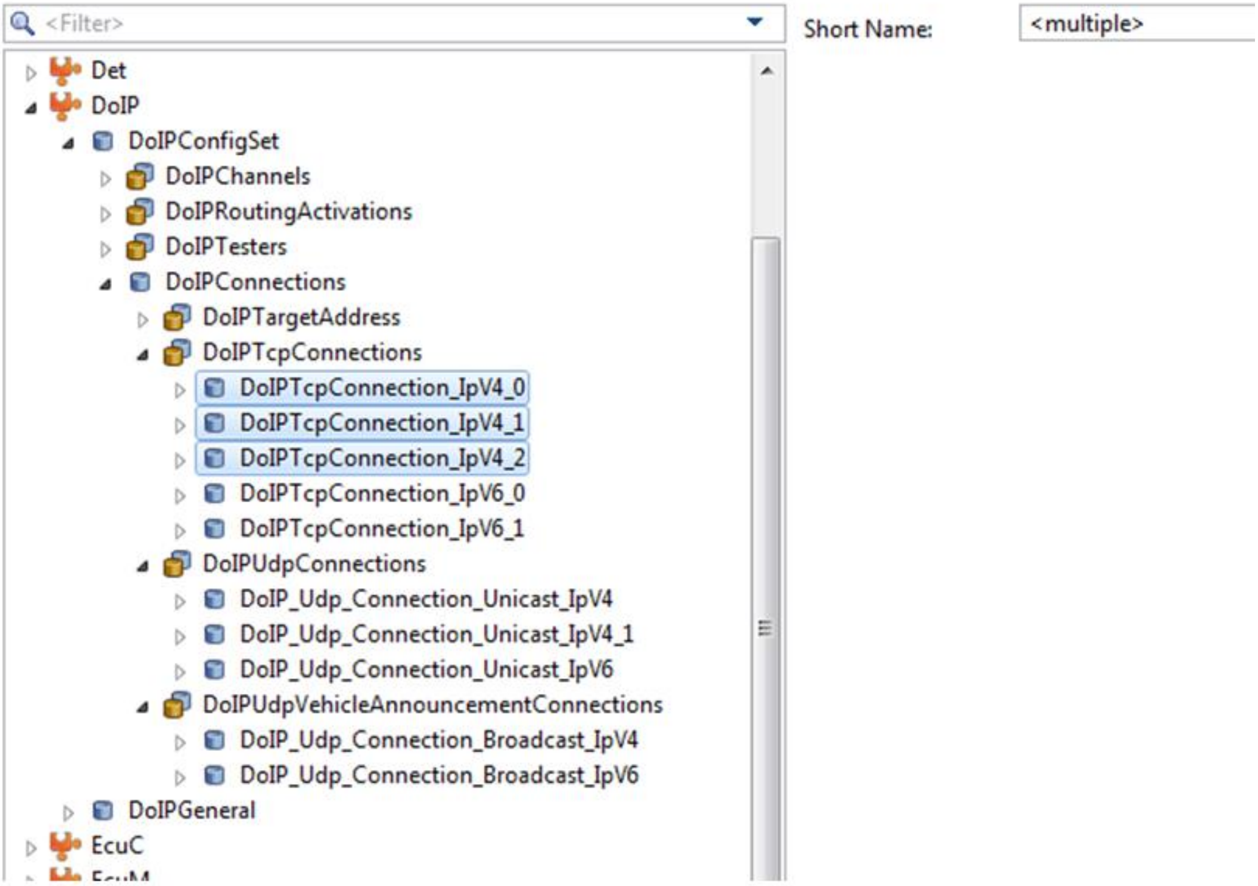
Remote Port not set: □

Figure 6-5 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 6-2).

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.



Short Name: <multiple>

- Det
- DoIP
  - DoIPConfigSet
    - DoIPChannels
    - DoIPRoutingActivations
    - DoIPTesters
    - DoIPConnections
      - DoIPTargetAddress
      - DoIPTcpConnections
        - DoIPTcpConnection_IpV4_0
        - DoIPTcpConnection_IpV4_1
        - DoIPTcpConnection_IpV4_2
        - DoIPTcpConnection_IpV6_0
        - DoIPTcpConnection_IpV6_1
      - DoIPUdpConnections
        - DoIP_Udp_Connection_Unicast_IpV4
        - DoIP_Udp_Connection_Unicast_IpV4_1
        - DoIP_Udp_Connection_Unicast_IpV6
      - DoIPUdpVehicleAnnouncementConnections
        - DoIP_Udp_Connection_Broadcast_IpV4
        - DoIP_Udp_Connection_Broadcast_IpV6
  - DoIPGeneral
- EcuC
- EcuM

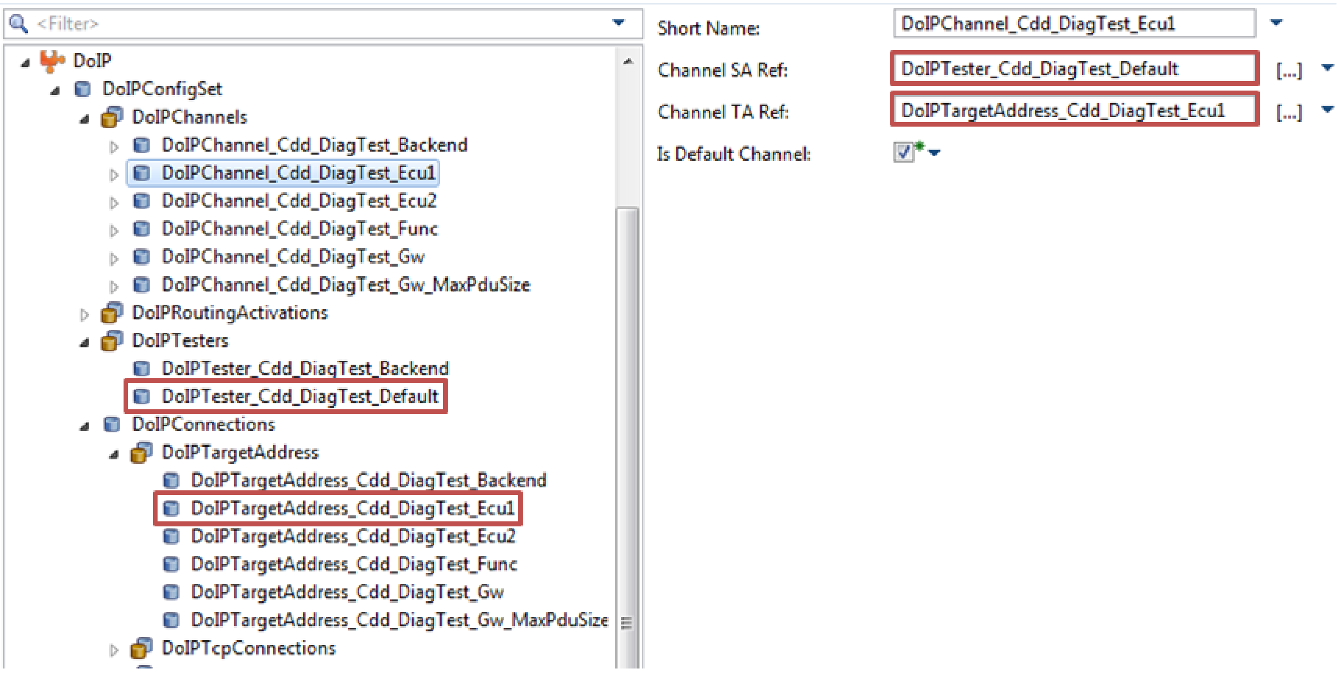
Figure 6-6 TCP connection configuration

6.2.1.3 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 6-7.

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.


<Filter>
DoIP
DoIPConfigSet
DoIPChannels
DoIPChannel_Cdd_DiagTest_Backend
DoIPChannel_Cdd_DiagTest_Ecu1
DoIPChannel_Cdd_DiagTest_Ecu2
DoIPChannel_Cdd_DiagTest_Func
DoIPChannel_Cdd_DiagTest_Gw
DoIPChannel_Cdd_DiagTest_Gw_MaxPduSize
DoIPRoutingActivations
DoIPTesters
DoIPTester_Cdd_DiagTest_Backend
DoIPTester_Cdd_DiagTest_Default
DoIPConnections
DoIPTargetAddress
DoIPTargetAddress_Cdd_DiagTest_Backend
DoIPTargetAddress_Cdd_DiagTest_Ecu1
DoIPTargetAddress_Cdd_DiagTest_Ecu2
DoIPTargetAddress_Cdd_DiagTest_Func
DoIPTargetAddress_Cdd_DiagTest_Gw
DoIPTargetAddress_Cdd_DiagTest_Gw_MaxPduSize
DoIPTcpConnections

Short Name: DoIPChannel_Cdd_DiagTest_Ecu1

Channel SA Ref: DoIPTester_Cdd_DiagTest_Default

Channel TA Ref: DoIPTargetAddress_Cdd_DiagTest_Ecu1

Is Default Channel: ☑

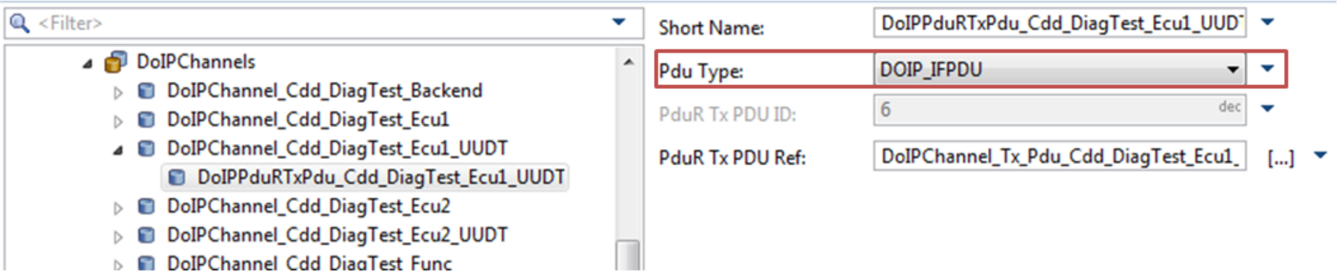
Figure 6-7 Channel configuration

6.2.1.3.1 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 6-8.


<Filter>
DoIPChannels
DoIPChannel_Cdd_DiagTest_Backend
DoIPChannel_Cdd_DiagTest_Ecu1
DoIPChannel_Cdd_DiagTest_Ecu1_UUDT
DoIPPduRTxPdu_Cdd_DiagTest_Ecu1_UUDT
DoIPChannel_Cdd_DiagTest_Ecu2
DoIPChannel_Cdd_DiagTest_Ecu2_UUDT
DoIPChannel_Cdd_DiagTest_Func

Short Name: DoIPPduRTxPdu_Cdd_DiagTest_Ecu1_UUD

Pdu Type: DOIP_IFPDU

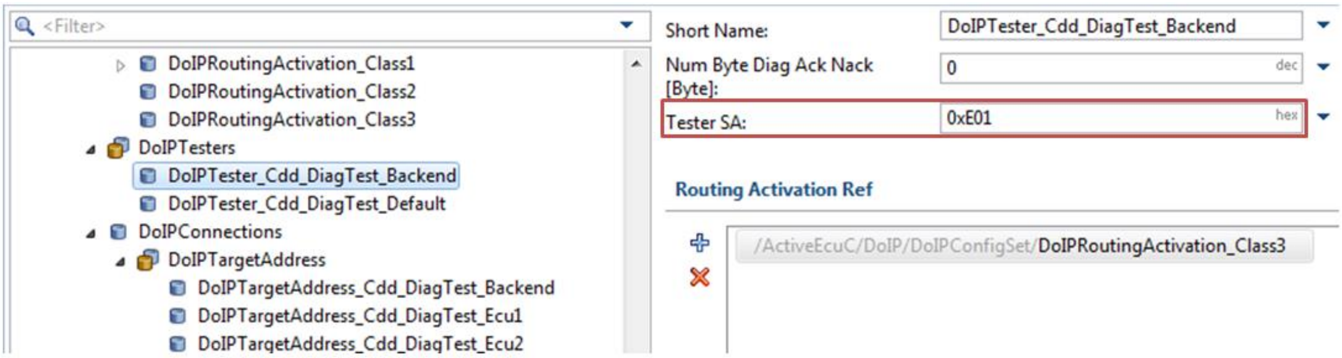
PduR Tx PDU ID: 6 dec

PduR Tx PDU Ref: DoIPChannel_Tx_Pdu_Cdd_DiagTest_Ecu1_

Figure 6-8 Channel UUDT configuration

6.2.1.4 Tester and Routing Activation configuration

Each tester is identified by a logical address (Figure 6-9). Additionally Routing Activation configurations have to be referenced by each tester.



DoIPRoutingActivation_Class1
DoIPRoutingActivation_Class2
DoIPRoutingActivation_Class3
DoIPTesters
DoIPTester_Cdd_DiagTest_Backend
DoIPTester_Cdd_DiagTest_Default
DoIPConnections
DoIPTargetAddress
DoIPTargetAddress_Cdd_DiagTest_Backend
DoIPTargetAddress_Cdd_DiagTest_Ecu1
DoIPTargetAddress_Cdd_DiagTest_Ecu2

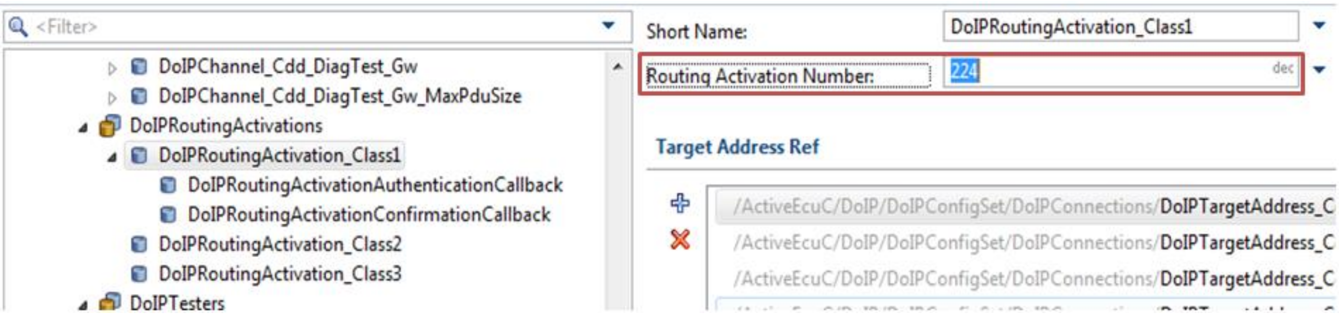
Short Name: DoIPTester_Cdd_DiagTest_Backend
Num Byte Diag Ack Nack [Byte]: 0 dec
Tester SA: 0xE01 hex

Routing Activation Ref

/ActiveEcuC/DoIP/DoIPConfigSet/DoIPRoutingActivation_Class3

Figure 6-9 Tester logical address

A Routing Activation is identified by a Routing Activation Number (Figure 6-10) which is received as “Activation type” (refer to Table 22 and 23 in [5]) in a Routing Activation Request message.



DoIPChannel_Cdd_DiagTest_Gw
DoIPChannel_Cdd_DiagTest_Gw_MaxPduSize
DoIPRoutingActivations
DoIPRoutingActivation_Class1
DoIPRoutingActivationAuthenticationCallback
DoIPRoutingActivationConfirmationCallback
DoIPRoutingActivation_Class2
DoIPRoutingActivation_Class3
DoIPTesters

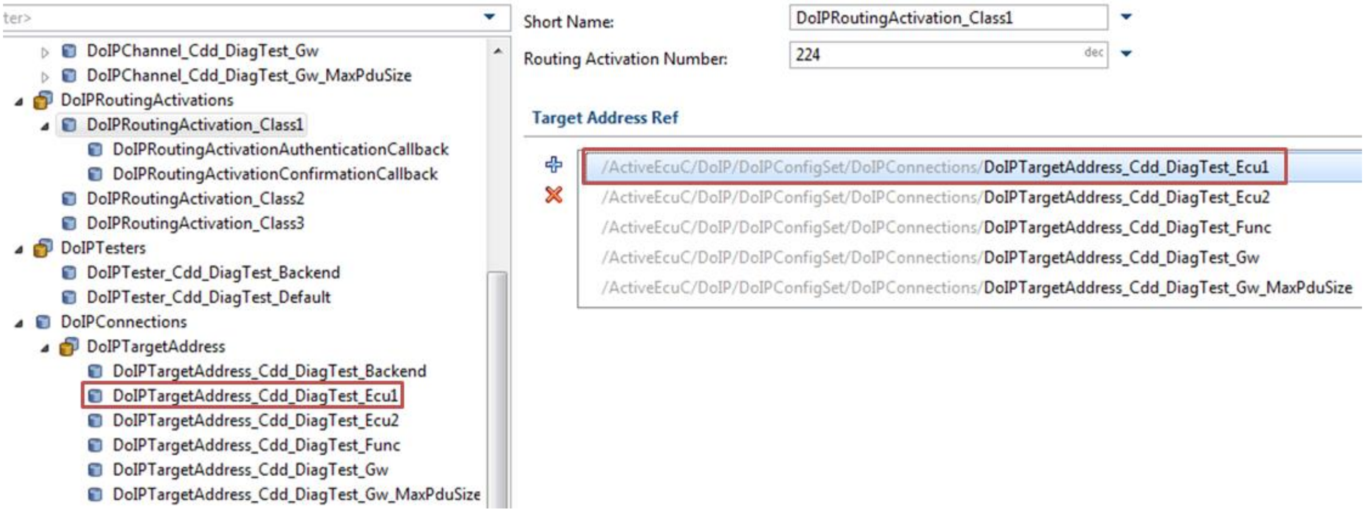
Short Name: DoIPRoutingActivation_Class1
Routing Activation Number: 224 dec

Target Address Ref

/ActiveEcuC/DoIP/DoIPConfigSet/DoIPConnections/DoIPTargetAddress_C
/ActiveEcuC/DoIP/DoIPConfigSet/DoIPConnections/DoIPTargetAddress_C
/ActiveEcuC/DoIP/DoIPConfigSet/DoIPConnections/DoIPTargetAddress_C

Figure 6-10 Routing Activation Number

If routing activation was successful all target addresses referenced by the Routing Activation (Figure 6-11) are accessible via diagnostic messages.



DoIPChannel_Cdd_DiagTest_Gw
DoIPChannel_Cdd_DiagTest_Gw_MaxPduSize
DoIPRoutingActivations
DoIPRoutingActivation_Class1
DoIPRoutingActivationAuthenticationCallback
DoIPRoutingActivationConfirmationCallback
DoIPRoutingActivation_Class2
DoIPRoutingActivation_Class3
DoIPTesters
DoIPTester_Cdd_DiagTest_Backend
DoIPTester_Cdd_DiagTest_Default
DoIPConnections
DoIPTargetAddress
DoIPTargetAddress_Cdd_DiagTest_Backend
DoIPTargetAddress_Cdd_DiagTest_Ecu1
DoIPTargetAddress_Cdd_DiagTest_Ecu2
DoIPTargetAddress_Cdd_DiagTest_Func
DoIPTargetAddress_Cdd_DiagTest_Gw
DoIPTargetAddress_Cdd_DiagTest_Gw_MaxPduSize

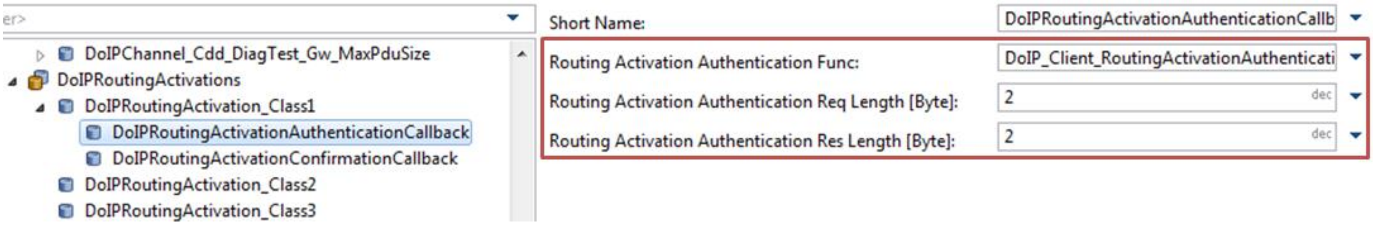
Short Name: DoIPRoutingActivation_Class1
Routing Activation Number: 224 dec

Target Address Ref

/ActiveEcuC/DoIP/DoIPConfigSet/DoIPConnections/DoIPTargetAddress_Cdd_DiagTest_Ecu1
/ActiveEcuC/DoIP/DoIPConfigSet/DoIPConnections/DoIPTargetAddress_Cdd_DiagTest_Ecu2
/ActiveEcuC/DoIP/DoIPConfigSet/DoIPConnections/DoIPTargetAddress_Cdd_DiagTest_Func
/ActiveEcuC/DoIP/DoIPConfigSet/DoIPConnections/DoIPTargetAddress_Cdd_DiagTest_Gw
/ActiveEcuC/DoIP/DoIPConfigSet/DoIPConnections/DoIPTargetAddress_Cdd_DiagTest_Gw_MaxPduSize

Figure 6-11 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 6-12).



er>

- DoIPChannel_Cdd_DiagTest_Gw_MaxPduSize
- DoIPRoutingActivations
  - DoIPRoutingActivation_Class1
    - DoIPRoutingActivationAuthenticationCallback
    - DoIPRoutingActivationConfirmationCallback
  - DoIPRoutingActivation_Class2
  - DoIPRoutingActivation_Class3

Short Name: DoIPRoutingActivationAuthenticationCallb

Routing Activation Authentication Func: DoIP_Client_RoutingActivationAuthenticati

Routing Activation Authentication Req Length [Byte]: 2 dec

Routing Activation Authentication Res Length [Byte]: 2 dec

Figure 6-12 Routing Activation Authentication/Confirmation callback

The “OEM specific” part can be separated in Authentication and Confirmation part of Routing Activation Request and Response. Figure 6-13 shows how length parameters are interpreted.

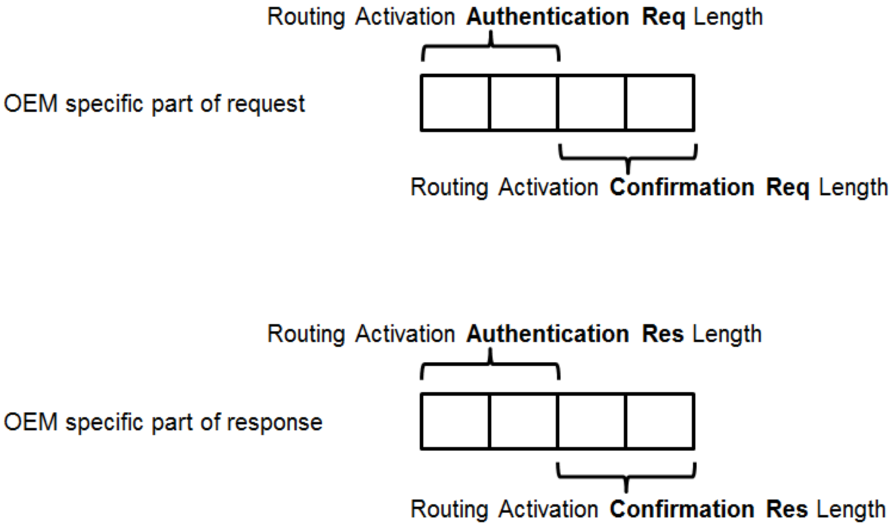

Routing Activation **Authentication** **Req** Length
OEM specific part of request
Routing Activation **Confirmation** **Req** Length
Routing Activation **Authentication** **Res** Length
OEM specific part of response
Routing Activation **Confirmation** **Res** Length

Figure 6-13 Routing Activation Authentication/Confirmation length parameter interpretation

6.2.1.5 Activation Line

The Activation Line can be used to enable or disable diagnostic communication. If the Activation Line is enabled DoIP requests the IP address assignment process on all DoIP dependent local addresses. If Activation Line is disabled DoIP releases the IP address assignments.

To switch the Activation Line state call `DoIP_ActivationLineSwitch()` described in 5.4.10. If function call is valid `<User>_DoIPActivationLineStateCallback()` described in 5.5.1.6 is called by DoIP to retrieve the Activation Line state. Set parameter `state` to `DOIP_ACTIVATION_LINE_INACTIVE` or `DOIP_ACTIVATION_LINE_ACTIVE` to indicate if Activation Line is inactive (diagnostic communication disabled) or active (diagnostic communication enabled).

Note
Communication of other users on the DoIP local addresses dependent on the Activation Line of DoIP. This means that no communication is possible if Activation Line is disabled. Users on other local addresses are not affected.

Caution
DoIP does not expect that other users request or release the IP address assignments on the DoIP dependent local addresses. In this case DET error `DOIP_E_UNEXPECTED_ASSIGNMENT` might be raised.

If `<User>_DoIPActivationLineStateCallback()` is not configured Activation Line must be enabled by default (parameter `DoIPActivationLineDefaultActive`). In this case IP assignments on local addresses are started automatically and will not be requested or released by DoIP.

6.2.2 Extended functionality

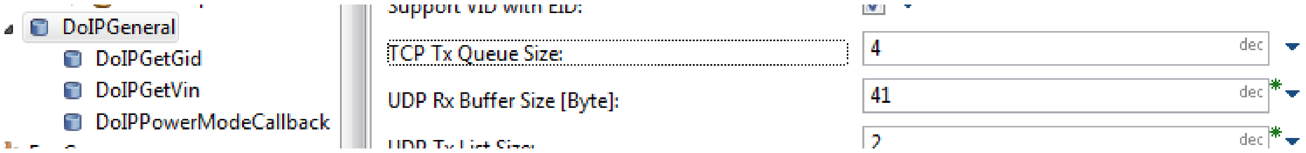
6.2.2.1 TcpTxQueue

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 6-14.



Support VID with EID:

TCP Tx Queue Size: 4 dec

UDP Rx Buffer Size [Byte]: 41 dec

UDP Tx List Size: 2 dec

Figure 6-14 Configure TcpTxQueue

If a Vector SoAd is used the parameter

`SoAd/SoAdConfig/SoAdSocketConnectionGroup/SoAdSocketConnection/SoAdTcpTxQueueSize`

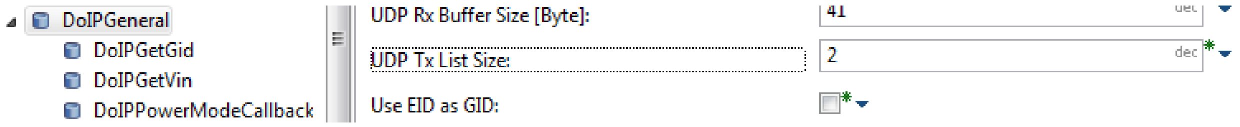
should be adapted, too. This queue handles transmission requests and their confirmations. In case of DoIP multiple and short messages may be sent in a short time. Therefore a bigger queue size is required. The size of DoIP TcpTxQueue can be used as approximate value.

To find this parameter the Configurator 5 “Find” view can be used.

6.2.2.2 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 6-15.



DoIPGeneral
DoIPGetGid
DoIPGetVin
DoIPPowerModeCallback

UDP Rx Buffer Size [Byte]: 41 dec

UDP Tx List Size: 2 dec

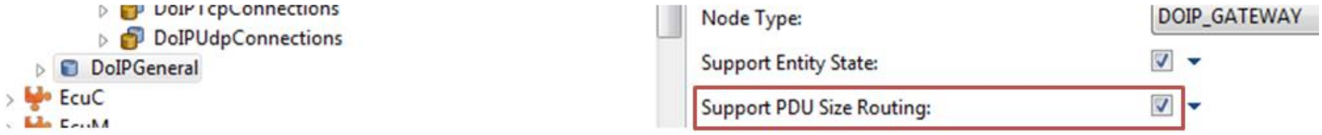
Use EID as GID: ☐

Figure 6-15 Configure UdpTxListSize

6.2.2.3 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 6-16) and at runtime via call to DoIP_EnablePduSizeRouting (disable via DoIP_DisablePduSizeRouting).



DoIPTcpConnections
DoIPUdpConnections
DoIPGeneral
EcuC
EcuM

Node Type: DOIP_GATEWAY

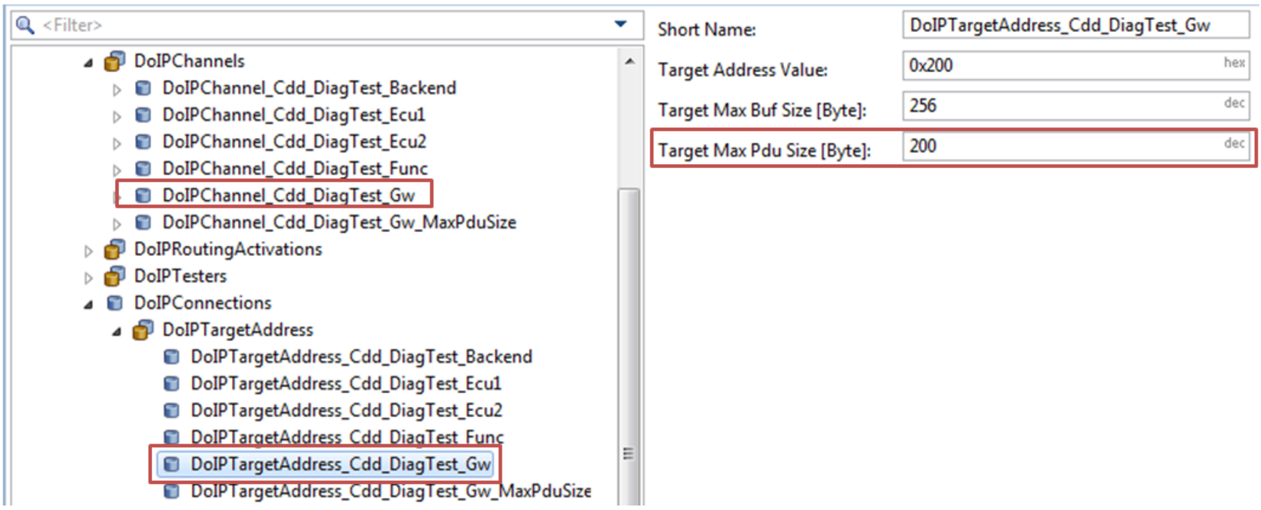
Support Entity State: ☑

Support PDU Size Routing: ☑

Figure 6-16 PDU Size Routing – enable support

Configure multiple channels and corresponding target addresses with same target address value but different maximum PDU sizes (Figure 6-17 and Figure 6-18).

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**. On reception of a length > 200 bytes and length <= 256 bytes user data are routed to DoIPTargetAddress_Cdd_DiagTest_**Gw_MaxPduSize**. If length of user data exceeds 256 bytes a “message to large” negative acknowledge is sent and user data are not routed.



<Filter>

- DoIPChannels
  - DoIPChannel_Cdd_DiagTest_Backend
  - DoIPChannel_Cdd_DiagTest_Ecu1
  - DoIPChannel_Cdd_DiagTest_Ecu2
  - DoIPChannel_Cdd_DiagTest_Func
  - DoIPChannel_Cdd_DiagTest_Gw
  - DoIPChannel_Cdd_DiagTest_Gw_MaxPduSize
- DoIPRoutingActivations
- DoIPTesters
- DoIPConnections
  - DoIPTargetAddress
    - DoIPTargetAddress_Cdd_DiagTest_Backend
    - DoIPTargetAddress_Cdd_DiagTest_Ecu1
    - DoIPTargetAddress_Cdd_DiagTest_Ecu2
    - DoIPTargetAddress_Cdd_DiagTest_Func
    - DoIPTargetAddress_Cdd_DiagTest_Gw
    - DoIPTargetAddress_Cdd_DiagTest_Gw_MaxPduSize

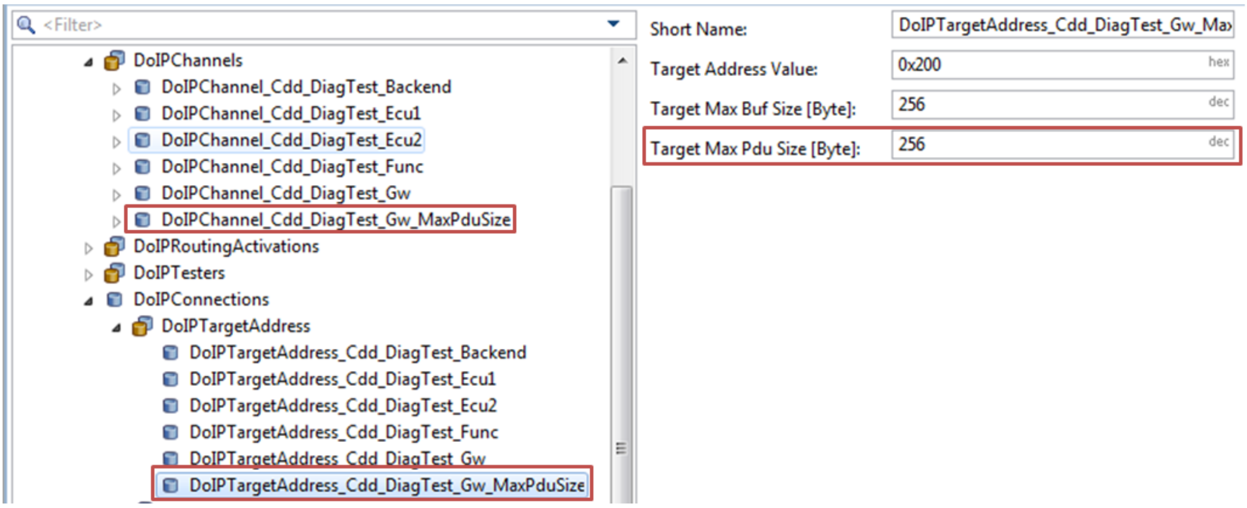
Short Name: DoIPTargetAddress_Cdd_DiagTest_Gw

Target Address Value: 0x200 hex

Target Max Buf Size [Byte]: 256 dec

Target Max Pdu Size [Byte]: 200 dec

Figure 6-17 PDU Size Routing – target address smaller size



<Filter>

- DoIPChannels
  - DoIPChannel_Cdd_DiagTest_Backend
  - DoIPChannel_Cdd_DiagTest_Ecu1
  - DoIPChannel_Cdd_DiagTest_Ecu2
  - DoIPChannel_Cdd_DiagTest_Func
  - DoIPChannel_Cdd_DiagTest_Gw
  - DoIPChannel_Cdd_DiagTest_Gw_MaxPduSize
- DoIPRoutingActivations
- DoIPTesters
- DoIPConnections
  - DoIPTargetAddress
    - DoIPTargetAddress_Cdd_DiagTest_Backend
    - DoIPTargetAddress_Cdd_DiagTest_Ecu1
    - DoIPTargetAddress_Cdd_DiagTest_Ecu2
    - DoIPTargetAddress_Cdd_DiagTest_Func
    - DoIPTargetAddress_Cdd_DiagTest_Gw
    - DoIPTargetAddress_Cdd_DiagTest_Gw_MaxPduSize

Short Name: DoIPTargetAddress_Cdd_DiagTest_Gw_Max

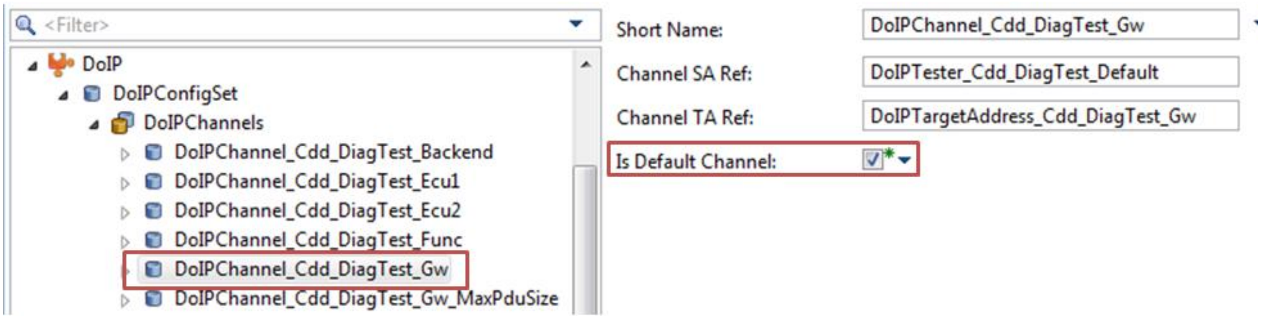
Target Address Value: 0x200 hex

Target Max Buf Size [Byte]: 256 dec

Target Max Pdu Size [Byte]: 256 dec

Figure 6-18 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 6-19).



<Filter>

- DoIP
  - DoIPConfigSet
    - DoIPChannels
      - DoIPChannel_Cdd_DiagTest_Backend
      - DoIPChannel_Cdd_DiagTest_Ecu1
      - DoIPChannel_Cdd_DiagTest_Ecu2
      - DoIPChannel_Cdd_DiagTest_Func
      - DoIPChannel_Cdd_DiagTest_Gw
      - DoIPChannel_Cdd_DiagTest_Gw_MaxPduSize

Short Name: DoIPChannel_Cdd_DiagTest_Gw

Channel SA Ref: DoIPTester_Cdd_DiagTest_Default

Channel TA Ref: DoIPTargetAddress_Cdd_DiagTest_Gw

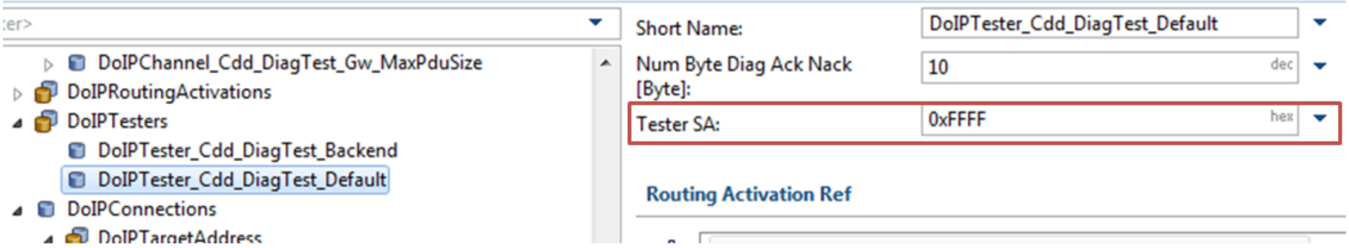
Is Default Channel: ☑

Figure 6-19 PDU Size Routing – default channel

6.2.2.4 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 6-20).



:er>
DoIPChannel_Cdd_DiagTest_Gw_MaxPduSize
DoIPRoutingActivations
DoIPTesters
DoIPTester_Cdd_DiagTest_Backend
DoIPTester_Cdd_DiagTest_Default
DoIPConnections
DoIPTargetAddress
Routing Activation Ref

Short Name: DoIPTester_Cdd_DiagTest_Default

Num Byte Diag Ack Nack [Byte]: 10 dec

Tester SA: 0xFFFF hex

Figure 6-20 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.

6.2.2.5 IPv4/6 and VLAN support

[1] AUTOSAR_SWS_DiagnosticOverIP.pdf specifies to support one Ethernet Interface with one IP address. But to support IPv4/6 parallel or multiple VLANs additional local IP addresses are required.

Vector DoIP supports multiple local IP addresses for TCP and UDP connections.

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

The configured IP address of each TCP and UDP connection is located in the corresponding SoAd socket connection group. Follow the referenced PDU in TCP or UDP connection to get corresponding SoAd PDU/Socket Route, socket connection and socket connection group.

Alive Check procedure is performed on specific local address only in case all TCP_DATA sockets of local address are already activated.

Alive Check procedure is performed on all connections independent of local address if same tester address is already activated.

A single instance of routing activation handler is implemented so parallel Routing Activation Requests are not supported on different local addresses.

Within UDP DoIP Entity Status Response message all TCP sockets, excluding reserve socket to decline additional testers of each configured address, are sent to represent the number of all TCP_DATA sockets.

6.2.2.6 Shutdown mechanism

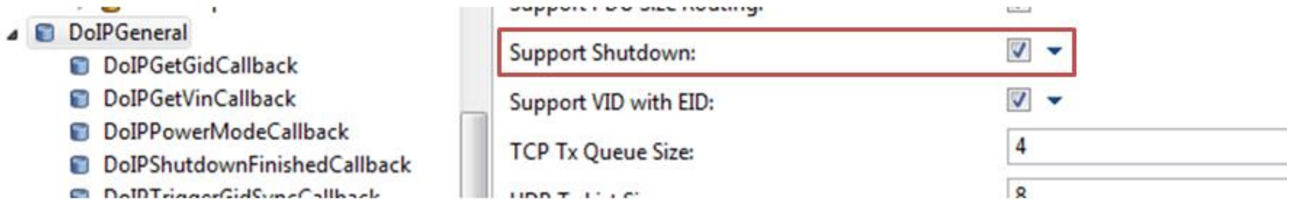
Vector DoIP supports a shutdown mechanism to prepare ECU and tester for an application/bootloader context transition. Therefore all open TCP sockets will be closed and DoIP module is placed into a specific shutdown state.

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 6-21 and to Figure 6-22 for callback configuration. SoAd configuration will be adapted via validation rules in configuration tool.



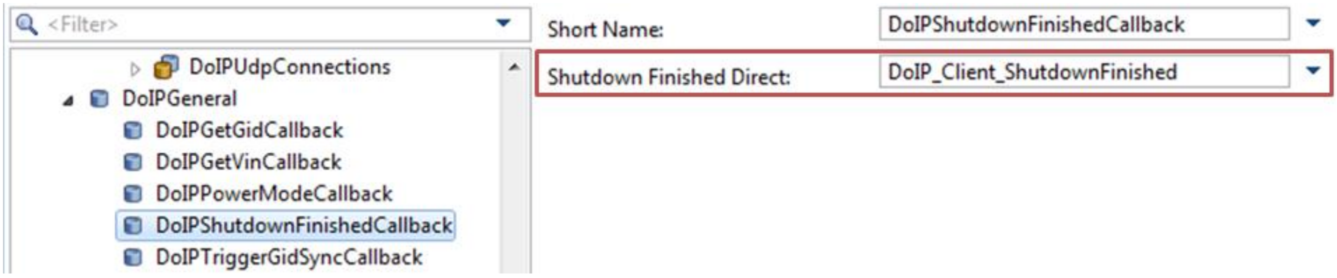
DoIPGeneral
- DoIPGetGidCallback
- DoIPGetVinCallback
- DoIPPowerModeCallback
- DoIPShutdownFinishedCallback

Support Shutdown: ☑

Support VID with EID: ☑

TCP Tx Queue Size: 4

Figure 6-21 Configuration example for DoIP Shutdown



<Filter>

- DoIPUdpConnections
- DoIPGeneral
  - DoIPGetGidCallback
  - DoIPGetVinCallback
  - DoIPPowerModeCallback
  - DoIPShutdownFinishedCallback
  - DoIPTriggerGidSyncCallback

Short Name: DoIPShutdownFinishedCallback

Shutdown Finished Direct: DoIP_Client_ShutdownFinished

Figure 6-22 Configuration example for DoIP Shutdown callback

6.2.2.7 OEM specific payload type

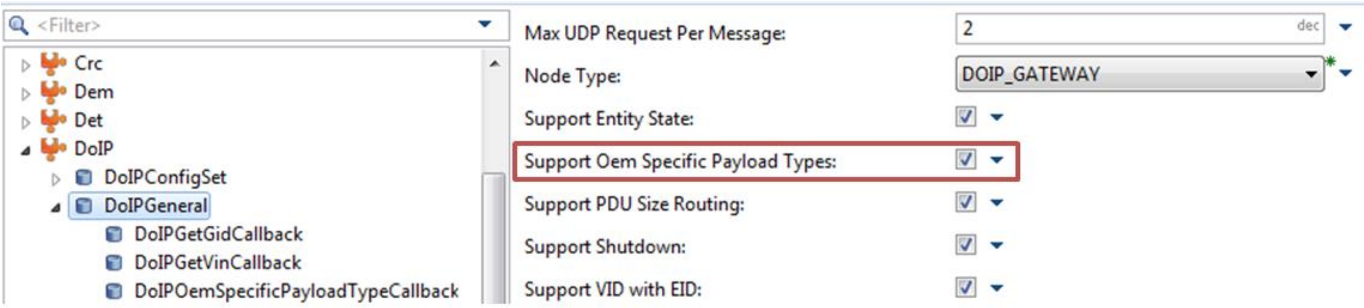
According to [5] DoIP supports manufacturer-specific payload types in range of 0xF000 to 0xFFFF but [1] does not provide an API to receive these payload types.

Vector supports a configurable callback to receive all unknown payload types over TCP and UDP. Via return value (refer to 5.5.1.9 for more information) user can indicate if payload type is known or unknown. If user indicates that payload type is unknown a generic DoIP header negative acknowledge message is sent as specified in [5] and [1]. Furthermore the callback provides parameters to send an optional response to the received message.

The feature can be enabled as described in Figure 6-23.

An additional buffer is needed to store the request in a linear buffer segment before forwarding to user in case of TCP. The same buffer is provided to the user to store the response. For UDP there is one buffer to store the user data of the optional response.

The buffer size and callback name configuration are described in Figure 6-24. If buffer sizes are set to 0 no response with user data can be sent (generic header only) and no user data can be received in case of TCP.



Max UDP Request Per Message: 2 dec

Node Type: DOIP_GATEWAY

Support Entity State: ☑

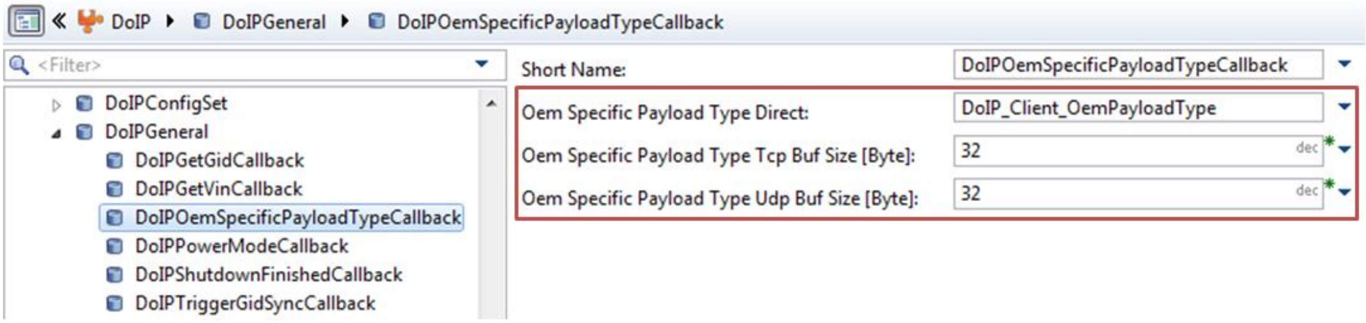
Support Oem Specific Payload Types: ☑

Support PDU Size Routing: ☑

Support Shutdown: ☑

Support VID with EID: ☑

Figure 6-23 OEM specific payload type configuration


« DoIP ▸ DoIPGeneral ▸ DoIPOemSpecificPayloadTypeCallback
<Filter>
DoIPConfigSet
DoIPGeneral
DoIPGetGidCallback
DoIPGetVinCallback
DoIPOemSpecificPayloadTypeCallback
DoIPPowerModeCallback
DoIPShutdownFinishedCallback
DoIPTriggerGidSyncCallback

Short Name: DoIPOemSpecificPayloadTypeCallback

Oem Specific Payload Type Direct: DoIP_Client_OemPayloadType

Oem Specific Payload Type Tcp Buf Size [Byte]: 32 dec

Oem Specific Payload Type Udp Buf Size [Byte]: 32 dec

Figure 6-24 OEM specific payload type buffer size and callback name configuration

Note
DoIP will forward all unknown payload types to user and is not restricted to the range defined for manufacturer-specific payload types (0xF000 to 0xFFFF).

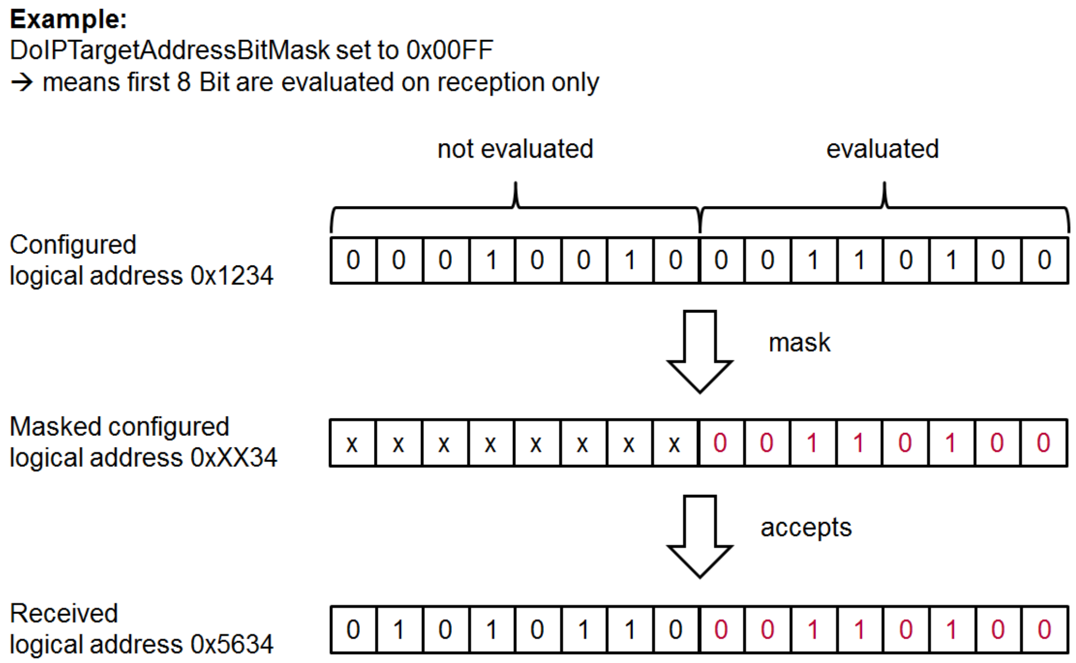
Caution
Since there is one global buffer to handle UDP frames a generic DoIP header negative acknowledge message is sent if buffer is in use (e.g. last response has not been sent yet).

6.2.2.8 Target Address Masking and Verification

6.2.2.8.1 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 6-25.



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

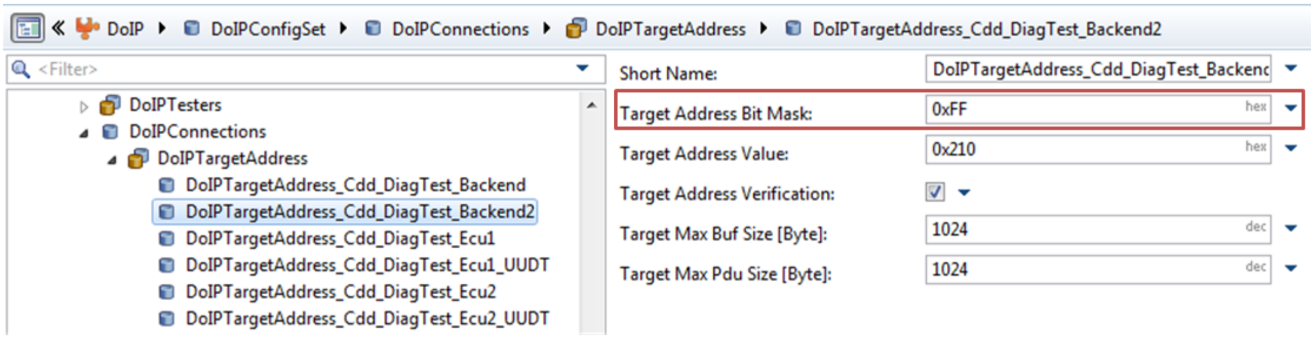
Configured logical address 0x1234: 0 0 0 1 0 0 1 0 0 0 1 1 0 1 0 0

Masked configured logical address 0xXX34: x x x x x x x x 0 0 1 1 0 1 0 0

Received logical address 0x5634: 0 1 0 1 0 1 1 0 0 0 1 1 0 1 0 0

Figure 6-25 Target Address Masking mechanism

The Masking can be configured on each channel (i.e. corresponding target address configuration) separately (Figure 6-26).



« DoIP ▸ DoIPConfigSet ▸ DoIPConnections ▸ DoIPTargetAddress ▸ DoIPTargetAddress_Cdd_DiagTest_Backend2
<Filter>
DoIPTesters
DoIPConnections
DoIPTargetAddress
DoIPTargetAddress_Cdd_DiagTest_Backend
DoIPTargetAddress_Cdd_DiagTest_Backend2
DoIPTargetAddress_Cdd_DiagTest_Ecu1
DoIPTargetAddress_Cdd_DiagTest_Ecu1_UUDT
DoIPTargetAddress_Cdd_DiagTest_Ecu2
DoIPTargetAddress_Cdd_DiagTest_Ecu2_UUDT

Short Name: DoIPTargetAddress_Cdd_DiagTest_Backenc

Target Address Bit Mask: 0xFF hex

Target Address Value: 0x210 hex

Target Address Verification: ☑

Target Max Buf Size [Byte]: 1024 dec

Target Max Pdu Size [Byte]: 1024 dec

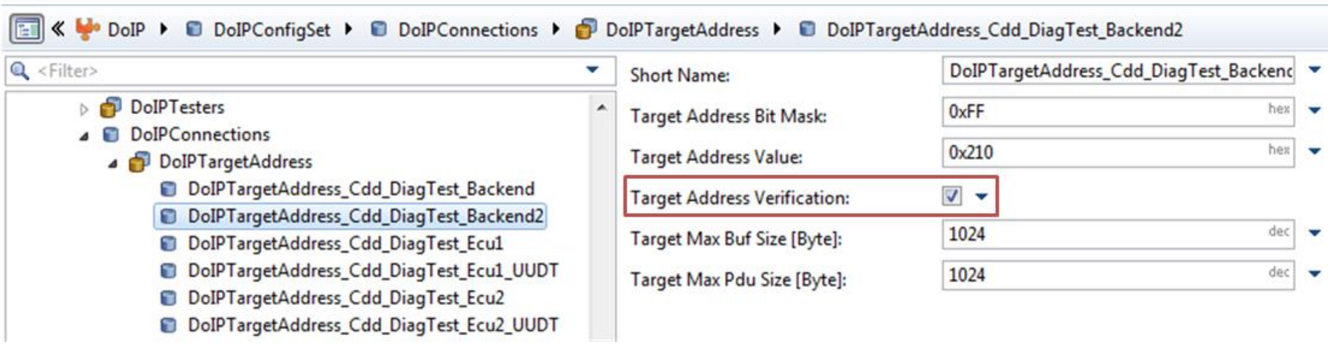
Figure 6-26 Target Address Masking configuration

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

6.2.2.8.2 Target Address Verification

Additionally a callback (5.5.1.10) 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 6-27. The callback name can be configured freely (Figure 6-28).


DoIP ▸ DoIPConfigSet ▸ DoIPConnections ▸ DoIPTargetAddress ▸ DoIPTargetAddress_Cdd_DiagTest_Backend2
<Filter>
DoIPTesters
DoIPConnections
DoIPTargetAddress
DoIPTargetAddress_Cdd_DiagTest_Backend
DoIPTargetAddress_Cdd_DiagTest_Backend2
DoIPTargetAddress_Cdd_DiagTest_Ecu1
DoIPTargetAddress_Cdd_DiagTest_Ecu1_UUDT
DoIPTargetAddress_Cdd_DiagTest_Ecu2
DoIPTargetAddress_Cdd_DiagTest_Ecu2_UUDT

Short Name: DoIPTargetAddress_Cdd_DiagTest_Backenc

Target Address Bit Mask: 0xFF hex

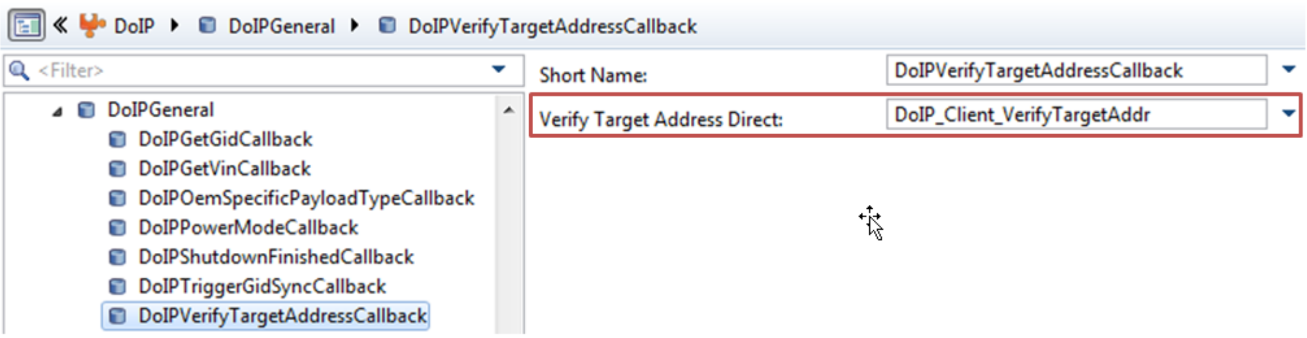
Target Address Value: 0x210 hex

Target Address Verification: ☑

Target Max Buf Size [Byte]: 1024 dec

Target Max Pdu Size [Byte]: 1024 dec

Figure 6-27 Target Address Verification target address configuration


DoIP ▸ DoIPGeneral ▸ DoIPVerifyTargetAddressCallback
<Filter>
DoIPGeneral
DoIPGetGidCallback
DoIPGetVinCallback
DoIPOemSpecificPayloadTypeCallback
DoIPPowerModeCallback
DoIPShutdownFinishedCallback
DoIPTriggerGidSyncCallback
DoIPVerifyTargetAddressCallback

Short Name: DoIPVerifyTargetAddressCallback

Verify Target Address Direct: DoIP_Client_VerifyTargetAddr

Figure 6-28 Target Address Verification callback configuration

6.2.2.9 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.

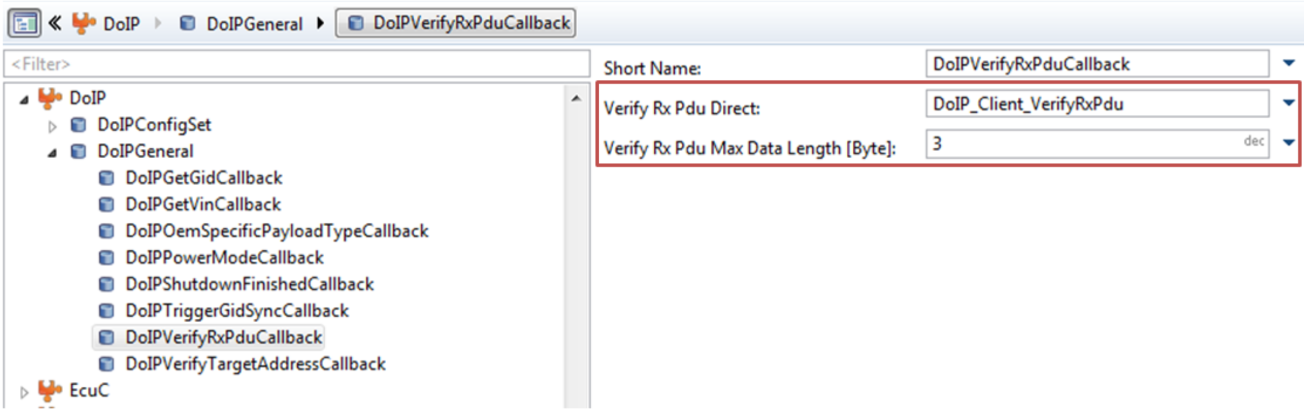
6.2.2.10 PDU reception verification

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 5.5.1.11) 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 6-29 shows how to configure the name of the callback and the maximum amount of PDU data which are forwarded via the callback.



Short Name: DoIPVerifyRxPduCallback

Verify Rx Pdu Direct: DoIP_Client_VerifyRxPdu

Verify Rx Pdu Max Data Length [Byte]: 3

Figure 6-29 PDU reception verification callback configuration

7 Glossary and Abbreviations

7.1 Glossary

Term	Description
Configurator 5	Generation tool for MICROSAR components

Table 7-1 Glossary

7.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
DEM	Diagnostic Event Manager
DET	Development Error Tracer
EAD	Embedded Architecture Designer
ECU	Electronic Control Unit
HIS	Hersteller Initiative Software
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
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
Tx	Transmission
TP	Transport Protocol (AUTOSAR API)
UDP	User Datagram Protocol
VIN	Vehicle Identification Number

Table 7-2 Abbreviations

8 Contact

Visit our website for more information on

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

www.vector.com