# MICROSAR Classic Diagnostic over IP for vehicle internal communication

Technical Reference

Version 6.0.0

| | |
|---|---|
| Authors | vismda, visjsb, viseje, visgyv |
| Status | Released |

# Document Information

## History

| Author | Date | Version | Remarks |
|---|---|---|---|
| vismda | 2019-07-18 | 1.0.0 | Creation of document |
| vismda | 2019-12-18 | 2.0.0 | Fixed some minor issues,<br>Message structure for DoIP light,<br>Configuration automation scripts,<br>Overwrite Logical Address API |
| viseje | 2021-07-09 | 2.0.1 | Update to current template |
| viseje | 2021-12-02 | 3.0.0 | Support UUDT |
| viseje | 2022-02-17 | 3.1.0 | Usage of DoIPInt_MemMap.h |
| viseje, vismda | 2022-05-11 | 4.0.0 | Product name updated to MICROSAR Classic,<br>OEM specific payload types,<br>Reception events |
| visgyv | 2023-05-11 | 5.0.0 | Provide channel ready for transmission information for Dcm,<br>Allow to transmit OEM specific payload types without payload |
| vismda, viseje | 2023-07-20 | 5.1.0 | Make DoIP Protocol Version configurable, Improved documentation for "Channel Ready Notification" |
| viseje | 2023-11-03 | 6.0.0 | Describe measures to detect unused TCP connections, Document nested calls of critical sections |

## Reference Documents

| No. | Source | Title | Version |
|---|---|---|---|
| [1] | AUTOSAR | Specification of Default Error Tracer | R4.2.2 |
| [2] | AUTOSAR | List of Basic Software Modules | R4.2.2 |
| [3] | ISO | ISO 13400-2 2012 | IS 2012 |
| [4] | ISO | ISO 13400-2 2019 | IS 2019 |
| [5] | Vector | TechnicalReference MICROSAR Classic DCM | See delivery |

**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 MICROSAR Classic BSW module DoIPInt.

| | | |
|---|---|---|
| **Supported Configuration Variants:** | pre-compile | |
| **Vendor ID:** | DOIPINT_VENDOR_ID | 30 decimal<br>(= Vector-Informatik, according to HIS) |
| **Module ID:** | DOIPINT_MODULE_ID | 255 decimal<br>(according to ref. [2]) |

The DoIPInt implements a transport protocol for diagnostic data over the Internet Protocol. It is derived from the DoIP module according to ISO Standard 13400 (see [3]) and adapted to fit the vehicle internal communication needs.

DoIP offers a lot of features which are not needed within a vehicle (e.g. routing activation of a tester). Therefore, DoIPInt implements only the generic header with the payload type “diagnostic message” according to [3]. All other messages (e.g. Generic DoIP header negative acknowledge) are not supported.

DoIP specifies that a tester always initiates the TCP connection (client) and an ECU waits for incoming TCP connections (server). DoIPInt implements both roles to make it usable in each node inside a vehicle (see Figure 1-1).

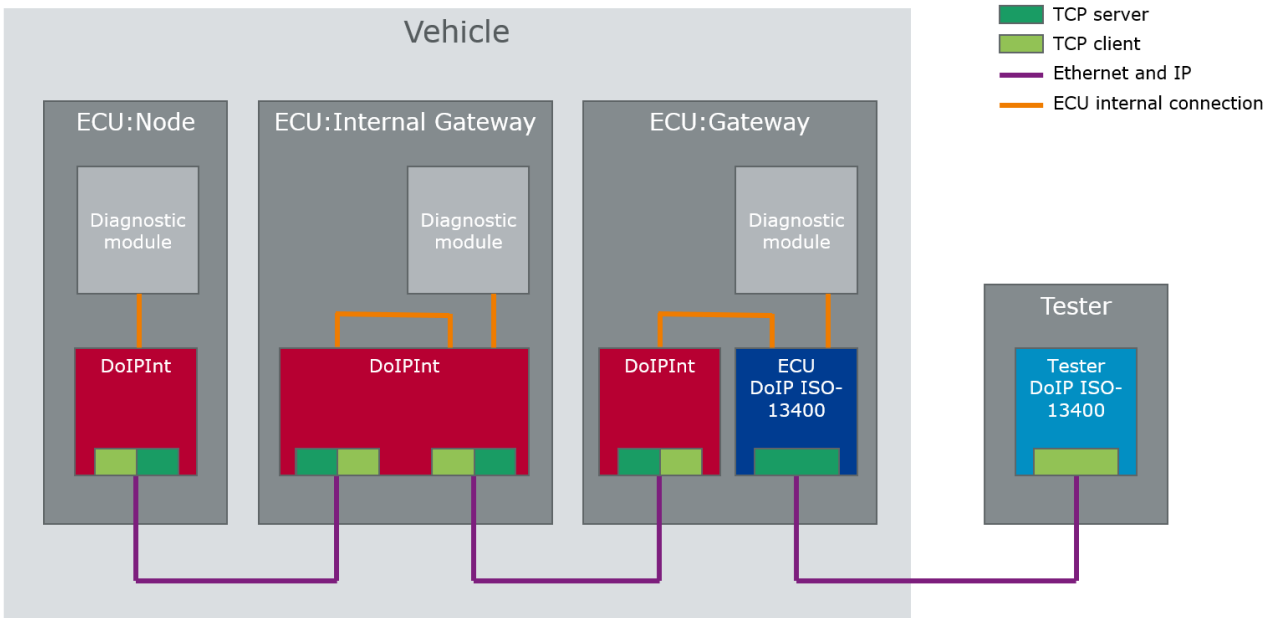


Figure 1-1 DoIPInt roles in an IP based vehicle network

## 1.1 Architecture Overview

The following figure shows where the DoIPInt is located in the MICROSAR Classic architecture.

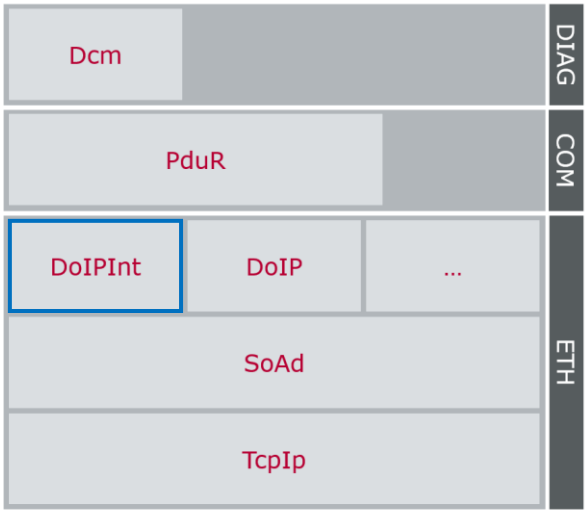


Figure 1-2 DoIPInt in MICROSAR Classic architecture

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

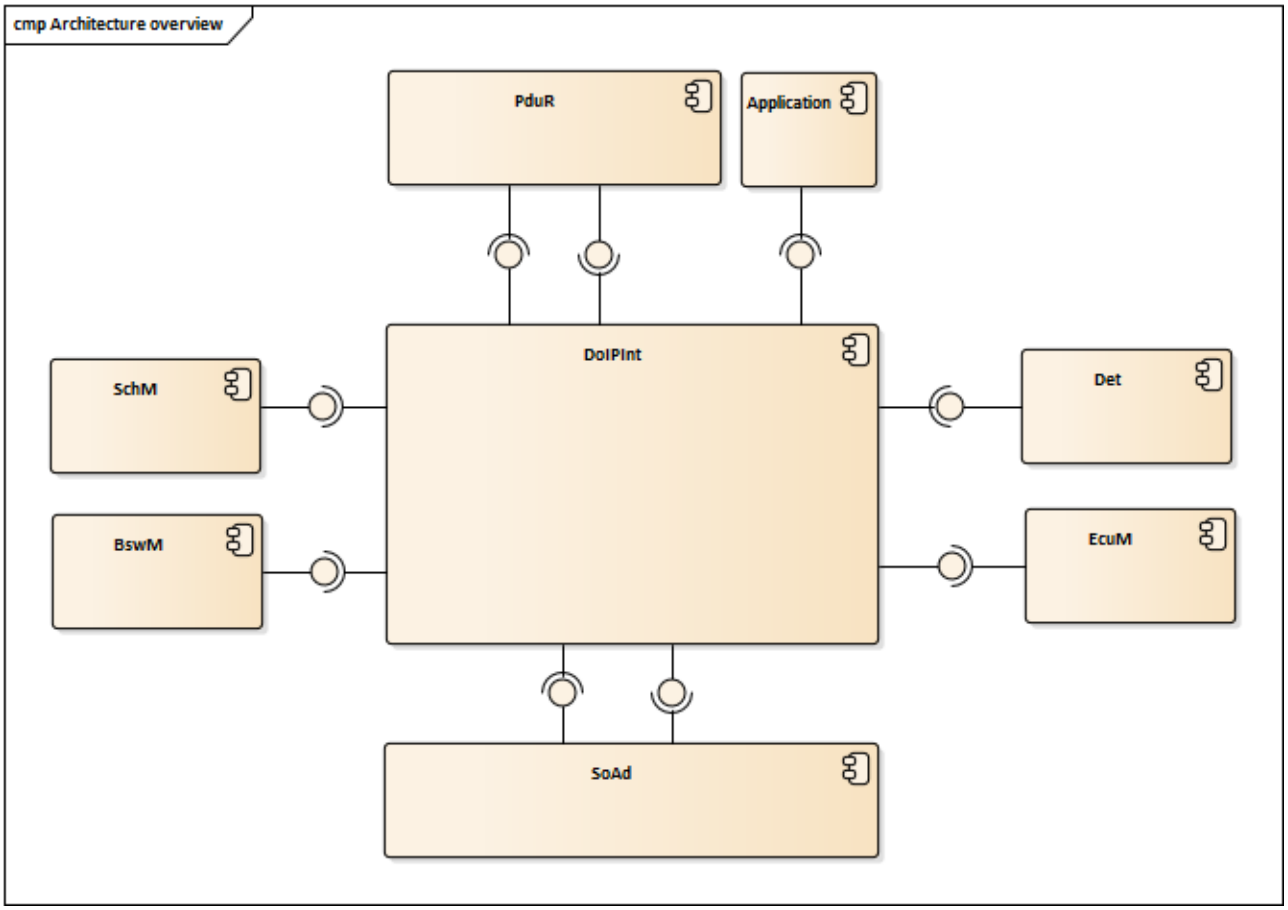


Figure 1-3 Interfaces to adjacent modules of the DoIPInt

# 2 Functional Description

## 2.1 Features

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

The following features are supported:

| Supported Features |
|---|
| Client TCP connection |
| Server TCP connection |
| Reopen of closed TCP connections |
| TCP Keep Alive mechanism |
| Transmission and reception of DoIP diagnostic messages over TCP connections |
| Cancellation of transmission and reception of DoIP diagnostic messages over TCP connections |
| Routing based on DoIP logical source and target address |
| Transmission queue for parallel transmission on same TCP connection |
| Reception retry on busy user |
| Configurable message structure |
| Overwritable logical addresses during runtime |
| UUDT over IF-API |
| Transmission and reception of OEM specific payload types |
| Notification of reception events |
| Provide channel ready to Dcm |
| Configurable DoIP protocol version |

Table 2-1 Supported Features

## 2.2 Initialization

The DoIPInt is initialized via a `DoIPInt_InitMemory()` call followed by a call to `DoIPInt_Init()`. A call to `DoIPInt_InitMemory()` reverts the initialization and a call to `DoIPInt_Init()` is required to initialize DoIPInt again.

## 2.3 States

DoIPInt provides an initialization state which is set by the services described in previous chapter.

## 2.4 Main Functions

The DoIPInt has one main function `DoIPInt_MainFunction()` which handles:

- The reopening of all closed TCP connections.
- The processing of all pending DoIPInt transmission requests.
- The processing of all pending DoIPInt reception retries.

## 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 [1], if development error reporting is enabled (i.e. pre-compile parameter `DOIPINT_DEV_ERROR_REPORT ==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 DoIPInt ID is 255.

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 | DoIPInt_Init() |
| 0x01 | DoIPInt_MainFunction() |
| 0x02 | DoIPInt_TpTransmit() |
| 0x03 | DoIPInt_TpCancelTransmit() |
| 0x04 | DoIPInt_TpCancelReceive() |
| 0x05 | DoIPInt_GetVersionInfo() |
| 0x06 | DoIPInt_SoConModeChg() |
| 0x07 | DoIPInt_SoAdTpCopyTxData() |
| 0x08 | DoIPInt_SoAdTpTxConfirmation() |
| 0x09 | DoIPInt_SoAdTpStartOfReception() |
| 0x0A | DoIPInt_SoAdTpCopyRxData() |
| 0x0B | DoIPInt_SoAdTpRxIndication() |
| 0x0C | DoIPInt_OverwriteLogicalAddresses() |
| 0x0D | DoIPInt_IfTransmit() |
| 0x0E | DoIPInt_IfCancelTransmit() |
| 0x0F | DoIPInt_TransmitOemSpecificPayloadType() |
| 0x10 | DoIPInt_CloseConnection() |

Table 2-2 Service IDs

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

| Error Code | Description |
|---|---|
| 0x00 | DOIPINT_E_NO_ERROR |
| 0x01 | DOIPINT_E_PARAM_CONFIG |

| Error Code | Description |
|---|---|
| 0x02 | DOIPINT_E_UNINIT |
| 0x03 | DOIPINT_E_PARAM |
| 0x04 | DOIPINT_E_PARAM_POINTER |

Table 2-3 Errors reported to DET

### 2.5.2 Production Code Error Reporting

DoIPInt does not report production code related errors.

# 3 Integration

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

## 3.1 Embedded Implementation

The delivery of the DoIPInt contains these source code files:

| File Name | Description | Integration Task |
|---|---|---|
| Appl_DoIPInt.c | Template source file for application | Adapt the template file. |
| Appl_DoIPInt.h | Template header file for application | Adapt the template file. |
| DoIPInt.c | Static source file | - |
| DoIPInt.h | Static header file | - |
| DoIPInt_Cbk.h | Static header file for callback functions | - |
| DoIPInt_Connection.c | Static source file of sub-module which handles connections | - |
| DoIPInt_Connection.h | Static header file of sub-module which handles connections | - |
| DoIPInt_DiagMsg.c | Static source file of sub-module which handles diagnostic messages | - |
| DoIPInt_DiagMsg.h | Static header file of sub-module which handles diagnostic messages | - |
| DoIPInt_Event.c | Static source file of sub-module which handles reception events | - |
| DoIPInt_Event.h | Static header file of sub-module which handles reception events | - |
| DoIPInt_GenHdr.c | Static source file of sub-module which handles generic header | - |
| DoIPInt_GenHdr.h | Static header file of sub-module which handles generic header | - |
| DoIPInt_LightHdr.c | Static source file of sub-module which handles messages with light header | - |
| DoIPInt_LightHdr.h | Static header file of sub-module which handles messages with light header | - |
| DoIPInt_OemSpecific.c | Static source file of sub-module which handles messages with OEM specific payload type | - |
| DoIPInt_OemSpecific.h | Static header file of sub-module which handles messages with OEM specific payload type | - |
| DoIPInt_Priv.h | Static header file for DoIPInt internal usage | - |
| DoIPInt_Rx.c | Static source file of sub-module which handles reception | - |
| DoIPInt_Rx.h | Static header file of sub-module which handles reception | - |
| DoIPInt_Tx.c | Static source file of sub-module which handles transmission | - |

| File Name | Description | Integration Task |
|---|---|---|
| DoIPInt_Tx.h | Static header file of sub-module which handles transmission | - |
| DoIPInt_TxQueue.c | Static source file of sub-module which handles transmission queue | - |
| DoIPInt_TxQueue.h | Static header file of sub-module which handles transmission queue | - |
| DoIPInt_Types.h | Static header file containing types | - |
| DoIPInt_Util.c | Static source file of sub-module which provides various utility | - |
| DoIPInt_Util.h | Static header file of sub-module which provides various utility | - |
| DoIPInt_Lcfg.c | Generated source file (e.g. RAM/ROM mapping tables) | - |
| DoIPInt_Lcfg.h | Generated header file | - |
| DoIPInt_Cfg.h | Generated header file for configuration parameter | - |
| DoIPInt_MemMap.h | Generated header file containing the memory sections of DoIPInt. | - |

Table 3-1 Implementation files

## 3.2 Critical Sections

All services and callbacks of DoIPInt may be called in interrupt or task context. 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 DoIPInt context are interrupt services that originate from physical bus events (Ethernet, CAN, LIN, FlexRay etc.).

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

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

> DOIPINT_EXCLUSIVE_AREA_0 is used whenever memory accesses must be protected from accesses of interrupting calls to services and callbacks of DoIPInt. 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 DoIPInt API functions (e.g. gateway use-case).

> Disable all Ethernet bus relevant interrupts if all modules calling DoIPInt 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.

**Caution**
Be aware that the DoIPInt calls its critical section nested. Please assert that this is supported by the implementation of the critical section.

## 3.3 Main Functions

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

**Caution**
Consider the mentioned main function expectations when using preemptive tasks (e.g. run all related modules in the same task to prevent interruption).

# 4 API Description

For an interface overview please see Figure 1-3.

## 4.1 Type Definitions

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

| Type Name | C-Type | Description | Value Range |
|---|---|---|---|
| DoIPInt_ChannelId Type | uint16 | DoIPInt unique identifier of the channel. | – |
| DoIPInt_Connectio nIdType | uint16 | DoIPInt unique identifier of the connection. | – |
| DoIPInt_EventType | uint8 | The result / status of the job | DOIPINT_EVENT_MSG_INV_PATTERN<br>Invalid pattern received (invalid protocol version). |
| | | | DOIPINT_EVENT_MSG_TOO_LARGE<br>The received message is too large for the configured OEM reception buffers or configured PduLengthType. |
| | | | DOIPINT_EVENT_MSG_INV_LENGTH<br>The received message length is not valid. |
| | | | DOIPINT_EVENT_MSG_RECEIVED_UP_TO_LE NGTH<br>The message has been received completely or up to DoIPIntEventMaxUserDataLength. |
| | | | DOIPINT_EVENT_MSG_DISCARDED<br>OEM specific message discarded as no reception buffers exist or all reception buffers are in use. |
| | | | DOIPINT_EVENT_DIAG_MSG_INV_REM_ADDR<br>The received message contains an invalid/unknown logical source address. |
| | | | DOIPINT_EVENT_DIAG_MSG_INV_LOCAL_AD DR<br>The received message contains an invalid/unknown logical target address. |
| | | | DOIPINT_EVENT_DIAG_MSG_DISCARDED<br>PduR_DoIPIntTpStartOfReception still fails after retry limit exceeded. |
| | | | DOIPINT_EVENT_DIAG_MSG_FORWARDED<br>The diagnostic message has been successfully forwarded to the user. |

Table 4-1 Type definitions

## 4.2 Services provided by DoIPInt

### 4.2.1 DoIPInt_Init

| Prototype | |
|---|---|
| `void DoIPInt_Init (const DoIPInt_ConfigType *ConfigPtr)` | |
| **Parameter** | |
| ConfigPtr [in] | Configuration structure for initializing the module. |
| **Return code** | |
| void | none |
| **Functional Description** | |
| Initializes component.<br>Service to initialize the module DoIPInt. It initializes all variables and sets the module state to initialized. | |
| **Particularities and Limitations** | |
| Interrupts are disabled and module main function is not triggered. | |
| Call context | |
| > TASK<br>> This function is Synchronous<br>> This function is Non-Reentrant | |

Table 4-2 DoIPInt_Init

### 4.2.2 DoIPInt_InitMemory

| Prototype | |
|---|---|
| `void DoIPInt_InitMemory (void)` | |
| **Parameter** | |
| void | none |
| **Return code** | |
| void | none |
| **Functional Description** | |
| Initializes *_INIT_*-variables.<br>Service to initialize module global variables at power up. This function initializes the variables in *_INIT_* sections. Used in case they are not initialized by the startup code. | |
| **Particularities and Limitations** | |
| Interrupts are disabled and module main function is not triggered. | |
| Call context | |
| > TASK<br>> This function is Synchronous<br>> This function is Non-Reentrant | |

Table 4-3 DoIPInt_InitMemory

### 4.2.3 DoIPInt_MainFunction

| Prototype | |
|---|---|
| `void DoIPInt_MainFunction (void)` | |
| **Parameter** | |
| void | none |
| **Return code** | |
| void | none |
| **Functional Description** | |
| Schedules the DoIPInt (Entry point for scheduling) and handles connection states. | |
| **Particularities and Limitations** | |
| - | |
| Call context | |
| > TASK<br>> This function is Synchronous<br>> This function is Non-Reentrant | |

Table 4-4 DoIPInt_MainFunction

### 4.2.4 DoIPInt_TpTransmit

| Prototype | |
|---|---|
| `Std_ReturnType DoIPInt_TpTransmit (PduIdType DoIPIntPduRTxId, const PduInfoType *DoIPIntPduRTxInfoPtr)` | |
| **Parameter** | |
| DoIPIntPduRTxId [in] | DoIPInt unique identifier of the PDU to be transmitted by the PduR. |
| DoIPIntPduRTxInfoPtr [in] | PDU information structure which contains the length 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 TP-PDU as diagnostic message.<br>- | |
| **Particularities and Limitations** | |
| - | |
| Call context | |
| > TASK\|ISR2<br>> This function is Non-Reentrant for same DoIPIntPduRTxId, Reentrant otherwise | |

Table 4-5 DoIPInt_TpTransmit

### 4.2.5 DoIPInt_TpCancelTransmit

| Prototype | |
|---|---|
| `Std_ReturnType` **`DoIPInt_TpCancelTransmit`** `(PduIdType DoIPIntPduRTxId)` | |
| **Parameter** | |
| DoIPIntPduRTxId [in] | DoIPInt unique identifier of the PDU to be transmitted by the PduR. |
| **Return code** | |
| Std_ReturnType | E_OK Request was accepted. |
| Std_ReturnType | E_NOT_OK Request was not accepted. |
| **Functional Description** | |
| Requests transmission cancellation of a TP-PDU.<br>- | |
| **Particularities and Limitations** | |
| - | |
| Call context | |
| > TASK\|ISR2<br>> This function is Non-Reentrant for same DoIPIntPduRTxId, Reentrant otherwise | |

Table 4-6 DoIPInt_TpCancelTransmit

### 4.2.6 DoIPInt_TpCancelReceive

| Prototype | |
|---|---|
| `Std_ReturnType` **`DoIPInt_TpCancelReceive`** `(PduIdType DoIPIntPduRRxId)` | |
| **Parameter** | |
| DoIPIntPduRRxId [in] | DoIPInt unique identifier of the PDU to be received by the PduR. |
| **Return code** | |
| Std_ReturnType | E_OK Request was accepted. |
| Std_ReturnType | E_NOT_OK Request was not accepted. |
| **Functional Description** | |
| Requests reception cancellation of a TP-PDU.<br>- | |
| **Particularities and Limitations** | |
| - | |
| Call context | |
| > TASK\|ISR2<br>> This function is Non-Reentrant for same DoIPIntPduRTxId, Reentrant otherwise | |

Table 4-7 DoIPInt_TpCancelReceive

### 4.2.7 DoIPInt_IfTransmit

| Prototype | |
|---|---|
| `Std_ReturnType DoIPInt_IfTransmit (PduIdType DoIPIntPduRTxId, const PduInfoType *DoIPIntPduRTxInfoPtr)` | |
| **Parameter** | |
| DoIPIntPduRTxId [in] | DoIPInt unique identifier of the PDU to be transmitted by the PduR. |
| DoIPIntPduRTxInfoPtr [in] | PDU information structure which contains the length and the 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 an IF-PDU as diagnostic message.<br>- | |
| **Particularities and Limitations** | |
| - | |
| Call context | |
| > TASK\|ISR2<br>> This function is Non-Reentrant for same DoIPIntPduRTxId, Reentrant otherwise | |

Table 4-8 DoIPInt_IfTransmit

### 4.2.8 DoIPInt_IfCancelTransmit

| Prototype | |
|---|---|
| `Std_ReturnType DoIPInt_IfCancelTransmit (PduIdType DoIPIntPduRTxId)` | |
| **Parameter** | |
| DoIPIntPduRTxId [in] | DoIPInt unique identifier of the PDU to be transmitted by the PduR. |
| **Return code** | |
| Std_ReturnType | E_OK Request was accepted. |
| Std_ReturnType | E_NOT_OK Request was not accepted. |
| **Functional Description** | |
| Requests transmission cancellation of an IF-PDU.<br>- | |
| **Particularities and Limitations** | |
| - | |
| Call context | |
| > TASK\|ISR2<br>> This function is Non-Reentrant for same DoIPIntPduRTxId, Reentrant otherwise | |

Table 4-9 DoIPInt_IfCancelTransmit

### 4.2.9 DoIPInt_TransmitOemSpecificPayloadType

| Prototype | |
|---|---|
| `Std_ReturnType` **`DoIPInt_TransmitOemSpecificPayloadType`** `(DoIPInt_ConnectionIdType ConnectionId, uint16 PayloadType, const PduInfoType *PayloadDataPtr)` | |
| **Parameter** | |
| ConnectionId [in] | DoIPInt 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.<br>- | |
| **Particularities and Limitations** | |
| - | |
| Call context | |
| > TASK\|ISR2<br>> This function is Synchronous<br>> This function is Non-Reentrant for same ConnectionId, Reentrant otherwise | |

Table 4-10 DoIPInt_TransmitOemSpecificPayloadType

### 4.2.10 DoIPInt_GetVersionInfo

| Prototype | |
|---|---|
| `void` **`DoIPInt_GetVersionInfo`** `(Std_VersionInfoType *Versioninfo)` | |
| **Parameter** | |
| Versioninfo [out] | Pointer to where to store the version information. |
| **Return code** | |
| void | none |
| **Functional Description** | |
| Returns the version information.<br>DoIPInt_GetVersionInfo() returns version information, vendor ID and module ID of the component. | |
| **Particularities and Limitations** | |
| -<br>Configuration Variant(s): DOIPINT_VERSION_INFO_API | |
| Call context | |
| > TASK\|ISR2<br>> This function is Synchronous | |

| |
|---|
| > This function is Reentrant |

Table 4-11 DoIPInt_GetVersionInfo

### 4.2.11 DoIPInt_OverwriteLogicalAddresses

| Prototype | |
|---|---|
| `Std_ReturnType` **`DoIPInt_OverwriteLogicalAddresses`** `(DoIPInt_ChannelIdType ChannelId, uint16 LogicalLocalAddress, uint16 LogicalRemoteAddress)` | |
| **Parameter** | |
| ChannelId [in] | DoIPInt unique identifier of the channel. |
| LogicalLocalAddress [in] | Local address of channel. |
| LogicalRemoteAddress [in] | Remote address of channel. |
| **Return code** | |
| Std_ReturnType | E_OK Request was accepted. |
| Std_ReturnType | E_NOT_OK Request was not accepted. |
| **Functional Description** | |
| Overwrites logical addresses until reinitialization.<br>- | |
| **Particularities and Limitations** | |
| - | |
| Call context | |
| > TASK\|ISR2<br>> This function is Synchronous<br>> This function is Non-Reentrant | |

Table 4-12 DoIPInt_OverwriteLogicalAddresses

### 4.2.12 DoIPInt_CloseConnection

| Prototype | |
|---|---|
| `Std_ReturnType` **`DoIPInt_CloseConnection`** `(DoIPInt_ConnectionIdType ConnectionId, boolean Abort)` | |
| **Parameter** | |
| ConnectionId [in] | DoIPInt unique identifier of the connection. |
| Abort [in] | Flag to close connection immediately.<br>[range: TRUE close immediately, FALSE close when Tx Queue is empty] |
| **Return code** | |
| Std_ReturnType | E_OK Request was accepted. |
| Std_ReturnType | E_NOT_OK Request was not accepted. |
| **Functional Description** | |
| Closes the requested connection.<br>- | |

| Particularities and Limitations |
|---|
| - |
| Call context |
| > TASK\|ISR2<br>> This function is Synchronous<br>> This function is Non-Reentrant for same ConnectionId, Reentrant otherwise |

Table 4-13 DoIPInt_CloseConnection

## 4.3 Services used by DoIPInt

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

| Component | API |
|---|---|
| BswM | BswM_DoIPInt_SetChannelReady |
| DET | Det_ReportError |
| IpBase | IpBase_GetUint8 |
| IpBase | IpBase_GetUint16 |
| IpBase | IpBase_GetUint32 |
| PduR | PduR_DoIPIntIfTxConfirmation |
| PduR | PduR_DoIPIntTpCopyTxData |
| PduR | PduR_DoIPIntTpTxConfirmation |
| PduR | PduR_DoIPIntTpStartOfReception |
| PduR | PduR_DoIPIntTpCopyRxData |
| PduR | PduR_DoIPIntTpRxIndication |
| SoAd | SoAd_OpenSoCon |
| SoAd | SoAd_CloseSoCon |
| SoAd | SoAd_TpTransmit |
| SoAd | SoAd_TpCancelTransmit |
| SoAd | SoAd_TpCancelReceive |
| VStdLib | VStdMemCpy |

Table 4-14 Services used by the DoIPInt

## 4.4 Callback Functions

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

### 4.4.1 DoIPInt_SoConModeChg

| Prototype |
|---|
| `void DoIPInt_SoConModeChg (SoAd_SoConIdType SoConId, SoAd_SoConModeType Mode)` |

| Parameter | |
|---|---|
| SoConId [in] | Socket connection index specifying the socket connection with mode change. |
| Mode [in] | New socket connection mode. |
| **Return code** | |
| void | none |
| **Functional Description** | |
| Receives notification for socket connection mode change.<br>- | |
| **Particularities and Limitations** | |
| - | |
| Call context | |
| > TASK<br>> This function is Synchronous<br>> This function is Non-Reentrant | |

Table 4-15 DoIPInt_SoConModeChg

### 4.4.2 DoIPInt_SoAdTpCopyTxData

| Prototype |
|---|
| `BufReq_ReturnType` **`DoIPInt_SoAdTpCopyTxData`** `(PduIdType DoIPIntSoAdTxPduId, PduInfoType *DoIPIntSoAdTxInfoPtr, RetryInfoType *RetryPtr, PduLengthType *AvailableDataPtr)` |

| Parameter | |
|---|---|
| DoIPIntSoAdTxPduId [in] | DoIPInt unique identifier of the PDU transmitted by the SoAd. |
| DoIPIntSoAdTxInfoPtr [in] | Provides the destination buffer and the number of bytes to be copied. |
| RetryPtr [in] | Parameter is not supported (NULL_PTR). |
| AvailableDataPtr [out] | Indicates the remaining number of bytes that are available to be copied. |
| **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** | |
| Copies data for transmission to provided buffer.<br>- | |
| **Particularities and Limitations** | |
| - | |
| Call context | |
| > TASK<br>> This function is Non-Reentrant | |

Table 4-16 DoIPInt_SoAdTpCopyTxData

### 4.4.3 DoIPInt_SoAdTpTxConfirmation

| Prototype | |
|---|---|
| `void DoIPInt_SoAdTpTxConfirmation (PduIdType DoIPIntSoAdTxPduId, Std_ReturnType Result)` | |
| **Parameter** | |
| DoIPIntSoAdTxPduId [in] | DoIPInt unique identifier of the PDU transmitted by the SoAd. |
| Result [in] | Result of the transmission. |
| **Return code** | |
| void | none |
| **Functional Description** | |
| Receives confirmation for transmitted PDU.<br>- | |
| **Particularities and Limitations** | |
| - | |
| Call context | |
| > TASK<br>> This function is Non-Reentrant | |

Table 4-17 DoIPInt_SoAdTpTxConfirmation

### 4.4.4 DoIPInt_SoAdTpStartOfReception

| Prototype | |
|---|---|
| `BufReq_ReturnType DoIPInt_SoAdTpStartOfReception (PduIdType DoIPIntSoAdRxPduId, PduInfoType *DoIPIntSoAdRxInfoPtr, PduLengthType TpSduLength, PduLengthType *AvailableDataPtr)` | |
| **Parameter** | |
| DoIPIntSoAdRxPduId [in] | DoIPInt unique identifier of the PDU to be received from the SoAd. |
| DoIPIntSoAdRxInfoPtr [in] | Parameter is not supported (NULL_PTR). |
| TpSduLength [in] | Length of the PDU. |
| AvailableDataPtr [out] | Indicates the remaining number of bytes that are available to be copied. |
| **Return code** | |
| BufReq_ReturnType | BUFREQ_OK Reception request was accepted. |
| BufReq_ReturnType | BUFREQ_E_NOT_OK Reception request was not accepted. |
| **Functional Description** | |
| Indicates start of reception of a PDU.<br>DoIPInt extracts the diagnostic messages from TCP stream. Therefore, function expects to be called with TpSduLength set to 0. | |
| **Particularities and Limitations** | |
| - | |
| Call context | |

| | |
|---|---|
| > TASK<br>> This function is Non-Reentrant | |

Table 4-18 DoIPInt_SoAdTpStartOfReception

### 4.4.5 DoIPInt_SoAdTpCopyRxData

| Prototype | |
|---|---|
| `BufReq_ReturnType` **`DoIPInt_SoAdTpCopyRxData`** `(PduIdType DoIPIntSoAdRxPduId, PduInfoType *DoIPIntSoAdRxInfoPtr, PduLengthType *AvailableDataPtr)` | |
| **Parameter** | |
| DoIPIntSoAdRxPduId [in] | DoIPInt unique identifier of the PDU to be received from the SoAd. |
| DoIPIntSoAdRxInfoPtr [in] | Provides the buffer and the number of bytes to be copied. |
| AvailableDataPtr [out] | Indicates the remaining number of bytes that are available to be copied. |
| **Return code** | |
| BufReq_ReturnType | BUFREQ_OK Data has been copied completely as requested. |
| BufReq_ReturnType | BUFREQ_E_NOT_OK Data has not been copied. Request failed. |
| **Functional Description** | |
| Copies data for reception from provided buffer.<br>- | |
| **Particularities and Limitations** | |
| - | |
| Call context | |
| > TASK<br>> This function is Non-Reentrant | |

Table 4-19 DoIPInt_SoAdTpCopyRxData

### 4.4.6 DoIPInt_SoAdTpRxIndication

| Prototype | |
|---|---|
| `void` **`DoIPInt_SoAdTpRxIndication`** `(PduIdType DoIPIntSoAdRxPduId, Std_ReturnType Result)` | |
| **Parameter** | |
| DoIPIntSoAdRxPduId [in] | DoIPInt unique identifier of the PDU to be received from the SoAd. |
| Result [in] | Result of the reception. |
| **Return code** | |
| void | none |
| **Functional Description** | |
| Receives indication for received PDU.<br>Since DoIPInt extracts the diagnostic messages from TCP stream this indication also indicates that socket connection is closed. | |

| Particularities and Limitations |
|---|
| - |
| Call context |
| > TASK<br>> This function is Non-Reentrant |

Table 4-20 DoIPInt_SoAdTpRxIndication

## 4.5 Configurable interfaces

### 4.5.1 Notifications

At its configurable interfaces the DoIPInt defines notifications that can be mapped to callback functions provided by other modules. The mapping is not statically defined by the DoIPInt but can be performed at configuration time. The function prototypes that can be used for the configuration have to match the appropriate function prototype signatures, which are described in the following sub-chapters.

#### 4.5.1.1 Appl_DoIPInt_ReceiveOemSpecificPayloadType

| Prototype | |
|---|---|
| `void` **`[DoIPIntOemPayloadRxCallback]`** `(DoIPInt_ConnectionIdType ConnectionId, uint16 PayloadType, const PduInfoType *PayloadDataPtr)` | |
| **Parameter** | |
| ConnectionId [in] | DoIPInt unique identifier of the connection. |
| PayloadType [in] | The received payload type. |
| PayloadDataPtr [in] | PDU information structure which contains the length and the payload data of the message. |
| **Return code** | |
| void | none |
| **Functional Description** | |
| Notifies about reception of an unknown payload type.<br>- | |
| **Particularities and Limitations** | |
| - | |
| Call context | |
| > TASK<br>> This function is Synchronous<br>> This function is Non-Reentrant for same ConnectionId, Reentrant otherwise | |

Table 4-21 Appl_DoIPInt_ReceiveOemSpecificPayloadType

#### 4.5.1.2 Appl_DoIPInt_Event

| Prototype |
|---|
| `Std_ReturnType` **`[DoIPIntEventCallback]`** `(DoIPInt_ConnectionIdType ConnectionId, DoIPInt_EventType Event, const PduInfoType *MsgDataPtr)` |

| Parameter | |
|---|---|
| ConnectionId [in] | DoIPInt unique identifier of the connection. |
| Event [in] | Event definition. |
| MsgDataPtr [in] | Pointer to entire DoIP message. |
| **Return code** | |
| Std_ReturnType | E_OK Handle event optimistic. |
| Std_ReturnType | E_NOT_OK Handle event pessimistic. |
| **Functional Description** | |
| Notifies about a DoIP protocol event or error.<br>- | |
| **Particularities and Limitations** | |
| - | |
| Call context | |
| > TASK<br>> This function is Synchronous<br>> This function is Non-Reentrant for same ConnectionId, Reentrant otherwise | |

Table 4-22 Appl_DoIPInt_Event

# 5 Configuration

## 5.1 Configuration Variants

The DoIPInt supports the configuration variants

> VARIANT-PRE-COMPILE

## 5.2 Configuration Design

Figure 5-1 provides an overview over the configuration design.

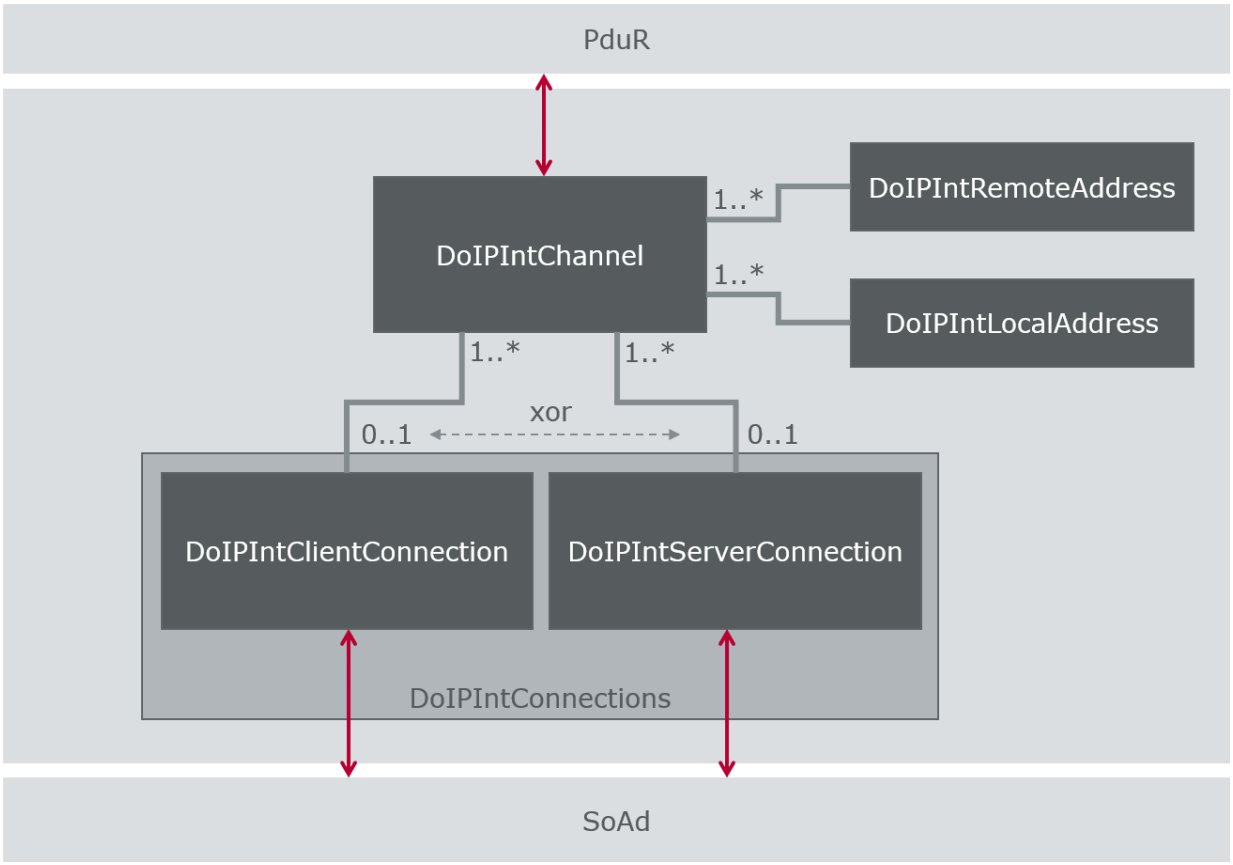


Figure 5-1 Configuration design overview

The DoIPIntConnections are the connections to the SoAd. Each DoIPIntClientConnection or DoIPIntServerConnection represents a separate TCP connection. Over these connections DoIPInt transmits or receives diagnostic messages. In case of DoIPIntClientConnection DoIPInt initiates the TCP connection. In case of DoIPIntServerConnection DoIPInt waits for an incoming TCP connection. It is possible to configure only DoIPIntClientConnection or DoIPIntServerConnection or both.

A DoIPIntChannel is the connection to the PduR. Over a DoIPIntChannel DoIPInt transmits or receives the user data of a diagnostic message (i.e. no header information). On transmission DoIPInt appends a DoIP header before the user data considering the

logical addresses of DoIPIntRemoteAddress and DoIPIntLocalAddress of the specific DoIPIntChannel. This diagnostic message is sent over the referenced DoIPIntClientConnection or DoIPIntServerConnection. On reception of a diagnostic message all DoIPIntChannels related to the DoIPIntClientConnection or DoIPIntServerConnection are searched to find the matching DoIPIntChannel. This is done by comparing the configured tuple of DoIPIntRemoteAddress and DoIPIntLocalAddress to the logical source and target address received in the DoIP header.

A DoIPIntRemoteAddress represents the logical address used in a diagnostic message as source address on reception and as target address on transmission.

A DoIPIntLocalAddress represents the logical address used in a diagnostic message as target address on reception and as source address on transmission.

For detailed configuration parameter descriptions please see the DoIPInt_bswmd.arxml file.

## 5.3 Configuration Automation

Enable the Script Task View within the DaVinci Configurator to see the automation scripts.

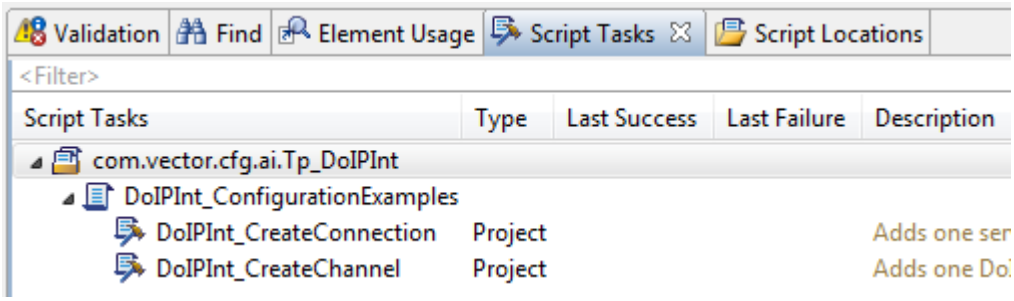


Figure 5-2 DoIPInt automation scripts

The DoIPInt provides two automation scripts which helps to easily create DoIPIntChannel, DoIPIntClientConnection and DoIPIntServerConnection.

The script task DoIPInt_CreateConnection adds one server/client connection for usage with DoIPInt to the configuration. This task configures DoIPInt, SoAd and EcuC for one DoIPInt connection. The connections/containers are created with a numerical postfix requested from the user. After the successful execution of the script task, manually assign the channels to the created connection in DoIPInt/DoIPIntConfigSet/DoIPIntChannel.

The script task DoIP_CreateChannel adds one DoIPInt channel to the configuration. Channel names/containers are postfixed with remote and/or local address. This task configures DoIPInt, PduR and EcuC for one DoIPInt channel. After the successful execution of the script task, manually assign the created channel to a connection in DoIPInt/DoIPIntConfigSet/DoIPIntChannel and complete the PduR routing path configuration.

## 5.4 Feature configuration

### 5.4.1 Configurable message structure

DoIPInt supports the header structure according to [3] (Figure 5-3) and an optimized “light” header structure (Figure 5-4). The structure is configurable globally (i.e. at runtime only one structure can be supported in parallel).

The configuration parameter `DoIPIntHeaderStructure` is used to select the required structure.

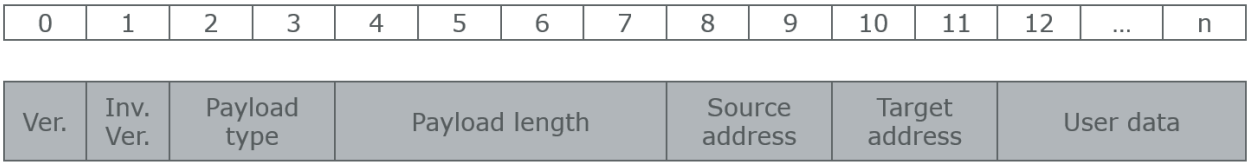


Figure 5-3 Header structure DoIP ISO 13400

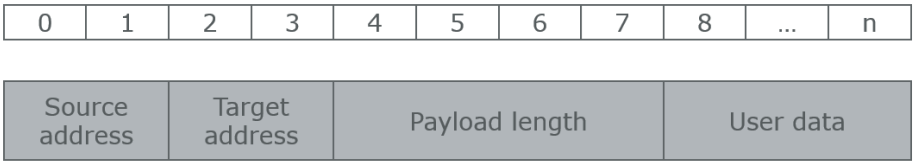


Figure 5-4 Header structure DoIP light

### 5.4.2 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 handled over TP-API to PduR, IF-API is used in case of UUDT.

Set the parameter `DoIPIntPduType` to “`DOIPINT_IFPDU`” to configure a channel for UUDT as shown in Figure 5-5.

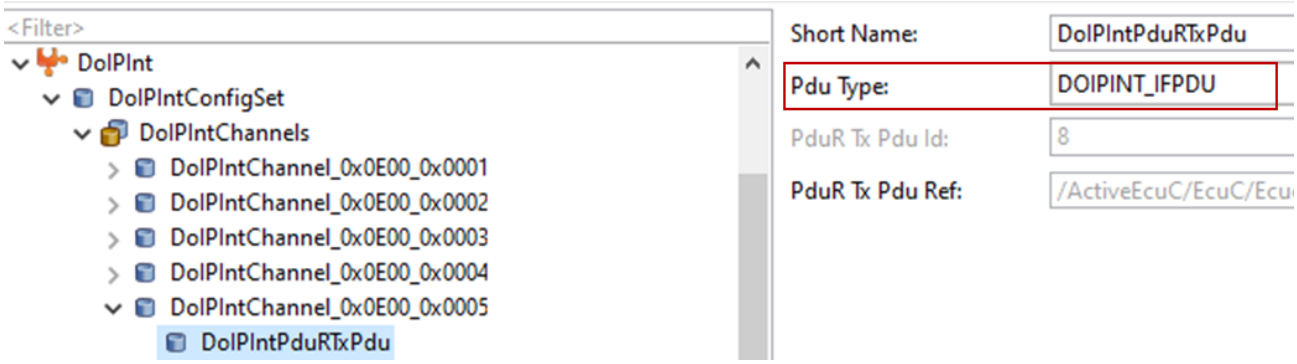


Figure 5-5 Configuration of an IF PDU

It is possible to configure the payload type for IF PDUs. This is done by setting the parameter `DoIPIntIfPayloadType` to the desired value (refer to Figure 5-6). By default, the diagnostic message payload type (0x8001) is used. This payload type is used for all IF PDUs while TP PDUs always use the diagnostic message payload type.

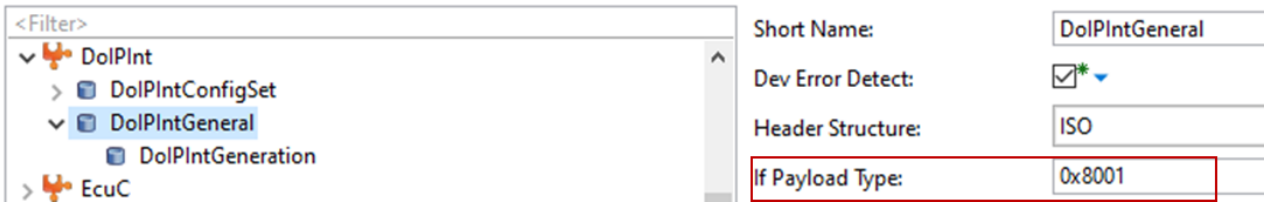


Figure 5-6 Configuration of the IF payload type

**Note**
A UUDT-only configuration is not supported. At least one channel using `DOIPINT_TPPDU` as `DoIPIntPduType` must be available in the configuration.

### 5.4.3 OEM specific payload types

As mentioned in chapter 1, DoIPInt only supports one message type. The message structure according to [3] allows to use other payload types. If required, the user can send other payload types by calling `DoIPInt_TransmitOemSpecificPayloadType()` (refer to chapter 4.2.9). To receive other payload types, the user configures and implements an `Appl_DoIPInt_ReceiveOemSpecificPayloadType()` as specified in chapter 4.5.1.1.

The feature itself can be enabled by adding the configuration container `DoIPInt/DoIPIntConfigSet/DoIPIntOemPayload`.

To enable OEM specific reception, it is required to configure the callback via `DoIPIntOemPayload/DoIPIntOemPayloadRxCallback`. An additional `DoIPIntOemPayloadRxBuffer` is required (refer to Figure 5-7). There is at maximum one buffer occupied by a connection at the same time.

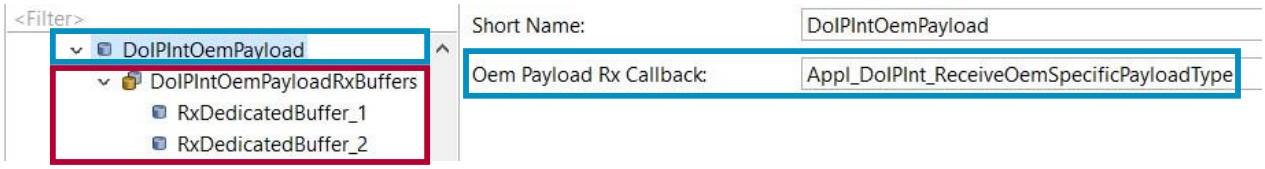


Figure 5-7 OEM specific payload type reception configuration

To be able to transmit OEM specific data `DoIPIntOemPayloadTxBuffer` must be configured as shown in Figure 5-8. Multiple `DoIPIntOemPayloadTxBuffer` may be used for a connection in parallel in case multiple transmissions are performed and the previous transmission is not yet confirmed completely.

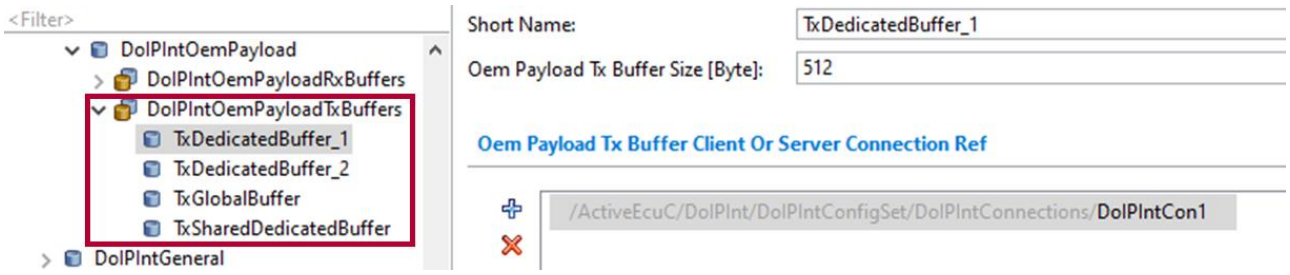


Figure 5-8 OEM specific payload type transmission configuration

For `DoIPIntOemPayloadRxBuffer` or `DoIPIntOemPayloadTxBuffer` one or several dedicated connections of an interface may be referenced via `DoIPIntOemPayloadRxBufferClientOrServerConnectionRef` or `DoIPIntOemPayloadTxBufferClientOrServerConnectionRef`. In case no connections are referenced the buffer is a global buffer (refer to the third buffer shown in Figure 5-9) and can be used for client or server 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 two buffers shown in Figure 5-9. In case a buffer references several connections, this is a shared (dedicated) buffer which can be used for client or server connections (refer to the fourth buffer shown in Figure 5-9). 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. `DoIPIntCon1` shown in Figure 5-9 will first of all try to reserve `TxDedicatedBuffer_1`. If the buffer is already in use, `TxSharedDedicatedBuffer` is checked for availability and at last `TxGlobalBuffer`. The buffers are released again when the message has been sent or forwarded to the user completely. `TxDedicatedBuffer_2` will not be used since it is a dedicated buffer for `DoIPIntCon2`.

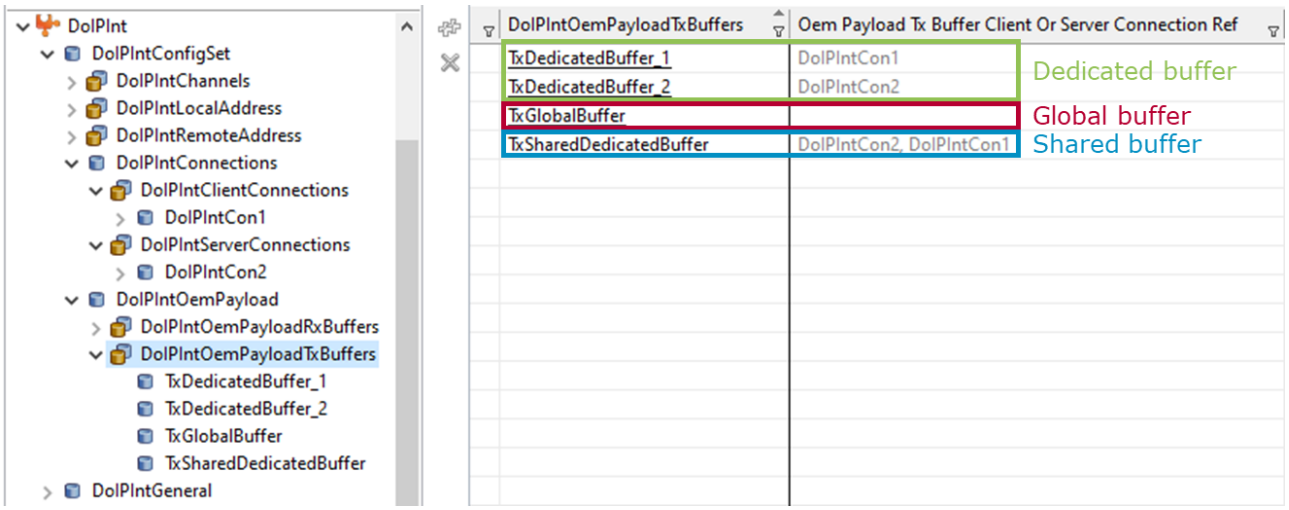


Figure 5-9 OEM specific payload type buffer configuration

If all buffers are in use on transmission, the transmission request is rejected at `DoIPInt_TransmitOemSpecificPayloadType()`.

If all buffers are in use on reception, the reception is retried until a suitable buffer is available or `DoIPIntGeneral/DoIPIntRxRetryTimeout` exceeded. If `DoIPIntGeneral/DoIPIntRxRetryTimeout` exceeded, the received message is dropped.

**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.

**Example**
The template files Appl_DoIPInt.c and Appl_DoIPInt.h implement an example for the usage of `DoIPInt_TransmitOemSpecificPayloadType()` and provide an example implementation of `Appl_DoIPInt_ReceiveOemSpecificPayloadType()`. For more details, refer to the content of the template files.

### 5.4.4 Reception events

There are multiple events or errors that may happen on reception of a message. A message may be malformed, for example, or it cannot be received since it is too large. There are also “positive” events like the successful reception of a message.

[3] specifies actions for these events. A socket may be closed or an acknowledge message may be sent, for example.

DoIPInt can be configured to report these events to the user. The callback to report the events is configured at `DoIPInt/DoIPIntGeneral/DoIPIntEventCallback/DoIPIntEventCallback`. The expected function declaration is specified at chapter 4.5.1.2.

The reported events and their descriptions can be found at `DoIPInt_EventType` in Table 4-1.

Together with the reported event, a specific amount of message data will be reported to the user. The reported data is at least the header information that is processed by DoIPInt. With `DoIPIntEventCallback/DoIPIntEventMaxUserDataLength` the amount of user data can be configured that shall be additionally passed to the user. It enables the user to decide how to react on an event based on the user data, too.

If the socket shall be closed based on the event, `DoIPInt_CloseConnection()` can be called by the user (refer to chapter 4.2.12).

The API to transmit and receive OEM specific payload types (refer to chapter 5.4.3) can be used to send messages like a generic header negative acknowledge message.

Depending on the return value of the callback function a different behavior for further message handling is defined. For details refer to Table 5-1.

| Event type | Event callback off | Event callback on, return E_OK | Event callback on, return E_NOT_OK |
|---|---|---|---|
| DOIPINT_EVENT_MSG_INV_PATTERN | Message reception failed, return BUFREQ_E_NOT_OK. | Continue message reception. | Skip message reception. |
| DOIPINT_EVENT_MSG_TOO_LARGE | Skip message reception. | Skip message reception. | Skip message reception. |
| DOIPINT_EVENT_MSG_INV_LENGTH | Skip message reception. | Skip message reception. | Skip message reception. |
| DOIPINT_EVENT_MSG_RECEIVED_UP_TO_LENGTH | Continue message reception. | Continue message reception. | Skip message reception. |
| DOIPINT_EVENT_MSG_DISCARDED | Skip message reception. | Skip message reception. | Skip message reception. |
| DOIPINT_EVENT_DIAG_MSG_INV_REM_ADDR | Skip message reception. | Skip message reception. | Skip message reception. |
| DOIPINT_EVENT_DIAG_MSG_INV_LOCAL_ADDR | Skip message reception. | Skip message reception. | Skip message reception. |
| DOIPINT_EVENT_DIAG_MSG_DISCARDED | Skip message reception. | Skip message reception. | Skip message reception. |
| DOIPINT_EVENT_DIAG_MSG_FORWARDED | Message reception finished. | Message reception finished. | Message reception finished. |

Table 5-1 Handling of the event callback return value

**Example**
The template files Appl_DoIPInt.c and Appl_DoIPInt.h implement an example for `Appl_DoIPInt_Event()`. For more details, refer to the content of the template files.

### 5.4.5 Channel Ready Notification for transmission

After rebooting of the ECU the Dcm may need to transmit outstanding messages. This is only possible if the TCP connection related to the channel is established. Since Dcm does not know when the establishment has succeeded, DoIPInt needs to notify the Dcm via `BswM_DoIPInt_SetChannelReady()` when the related ComM channel is ready to transmit messages.

The boolean parameter `/MICROSAR/DoIPInt/DoIPIntConfigSet/DoIPIntChannel/DoIPIntChannelReadyNotification` enables the channel ready notification (see Figure 5-10).

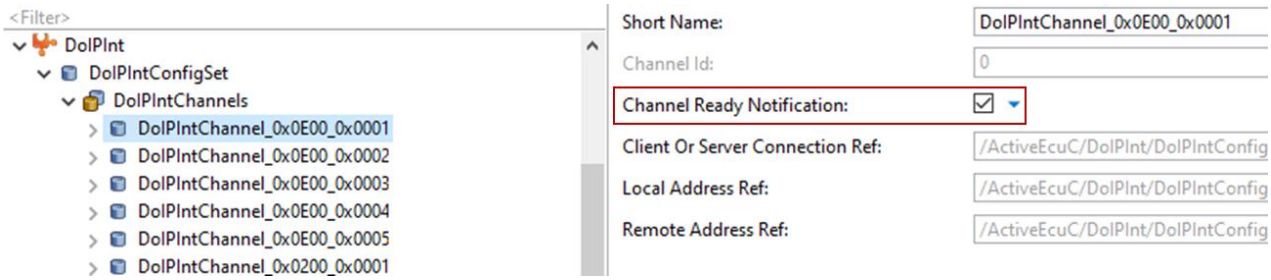


Figure 5-10 DoIPInt channel ready notification configuration

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 [5].

Additionally, it is necessary to configure a BswM rule (`BswMModeRequestPort` = “DoIPInt Set Channel Ready”, `BswMAction` = “Dcm Set Channel Ready”) within DaVinci Configurator (see Figure 5-11) 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`”.

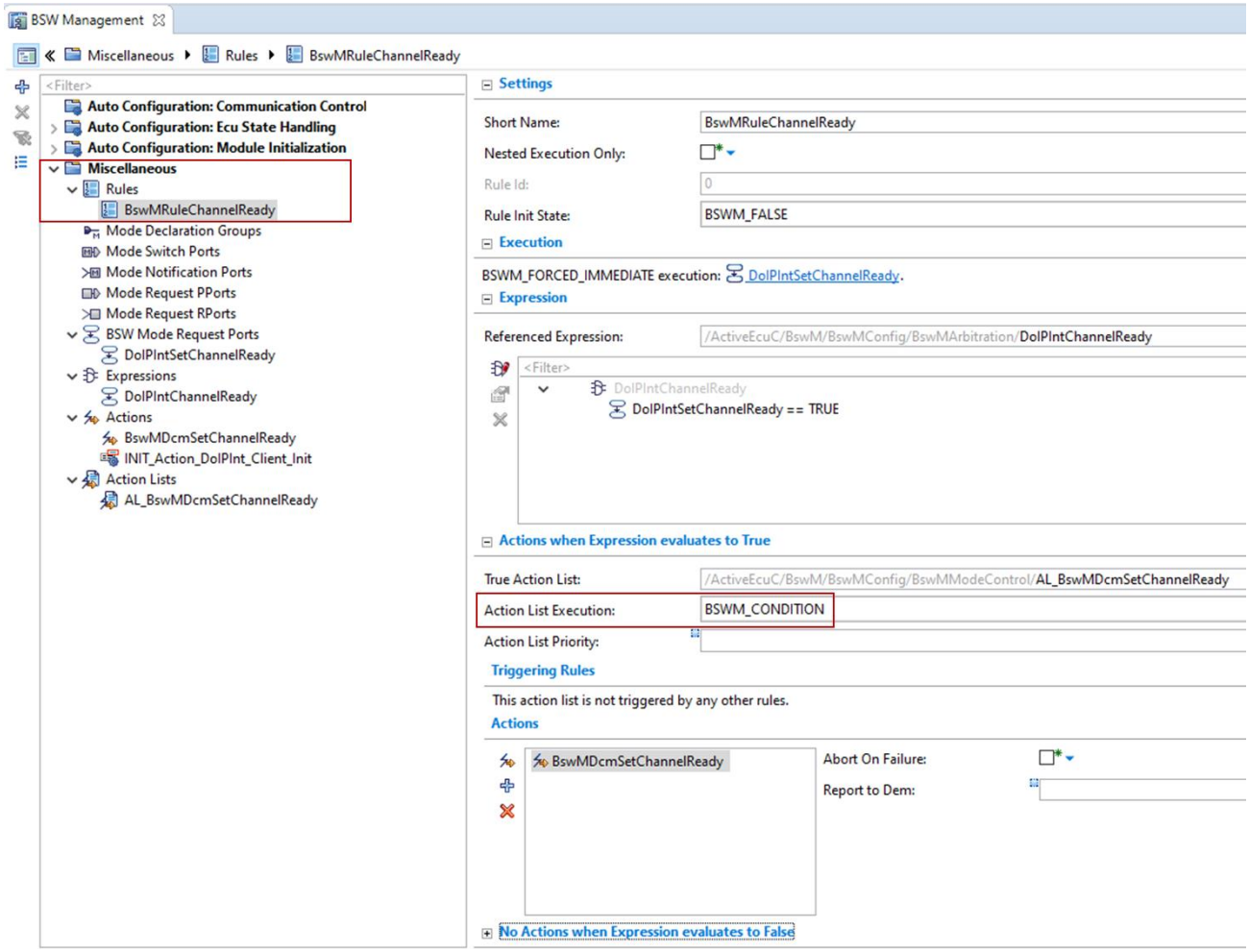


Figure 5-11 BswM rule to configure the channel ready feature

**Note**
DoIPInt calls this notification whenever a TCP connection is established. This may result in multiple calls if multiple connections are supported or if a connection is established again.

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

### 5.4.6 Configurable DoIP protocol version

This chapter only applies if the header structure as defined in [3] ISO 13400-2 2012 (refer to chapter 5.4.1) is used.

As DoIPInt implements only parts of the ISO 13400-2, it is compatible to the latest ISO 13400-2 versions (please refer to [3] ISO 13400-2 2012 and [4] ISO 13400-2 2019).

Nevertheless, it may make sense to use the same DoIP protocol version vehicle-internally and externally. Therefore, the DoIP protocol version which is used on transmission inside the DoIP generic header and accepted on reception is configurable (refer to Figure 5-12).

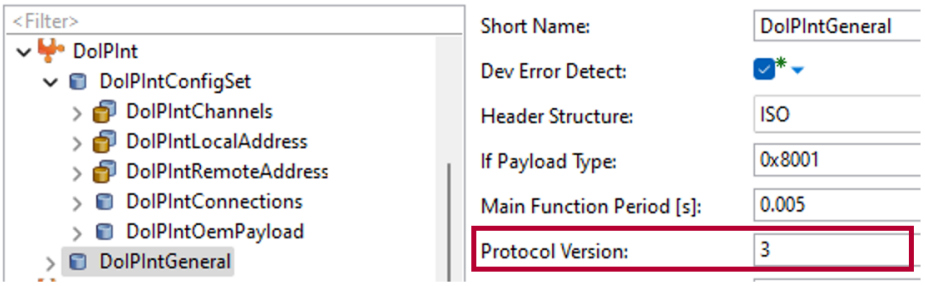


Figure 5-12 DoIP protocol version configuration

**Note**
The DoIP *inverse* protocol version is automatically derived from the configured DoIP protocol version. There is no need to configure it separately.

## 5.5 Configuration Hints

### 5.5.1 Detection of unused TCP connections

While DoIP protocol according to ISO specifies the routing activation and alive check mechanism to detect unused connections (e.g. outdated by a tester/client reboot), there is currently no predefined way to detect such connections in case of DoIPInt. However, several mechanisms can be applied to detect unused TCP connections which are described in this chapter. This chapter uses a reboot as exemplary cause for unused TCP connections, but the mechanisms may of course also be used in other scenarios that lead to unused TCP connections.

#### 5.5.1.1 Ensure that TCP connections are closed before a shutdown/reboot

It can be ensured by the user of the DoIPInt that connections are closed by calling `DoIPInt_CloseConnection()` before a reboot is performed as shown in Figure 5-13. This is very easily possible since no protocol extension is required. On the other hand, it does not work for “unexpected” reboots.

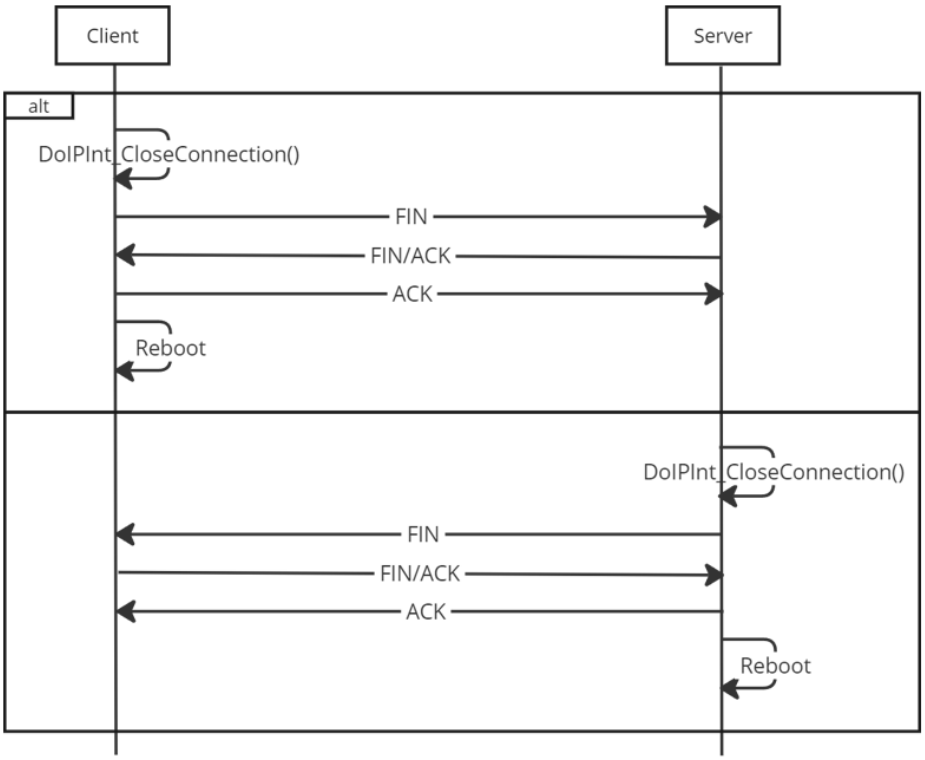


Figure 5-13 Close connection before reboot via DoIPInt_CloseConnection() API

### 5.5.1.2 Use TCP Keep-Alive

The TCP Keep-Alive mechanism can also be used to detect unused TCP connections. Keep-Alive must be enabled on client and server sockets since a reboot may occur on both entities and configuration is done in the TcpIp or Socket Adaptor component. Refer to Figure 5-14 to see how this mechanism works.

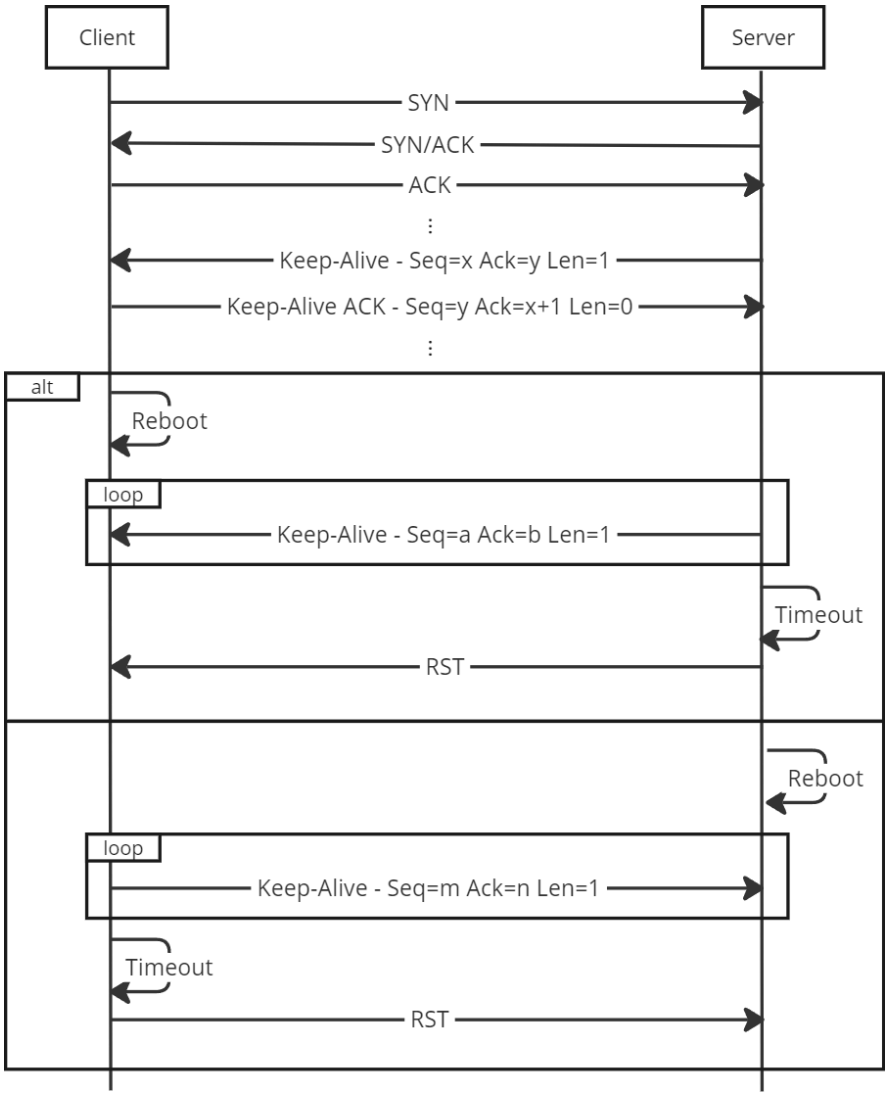


Figure 5-14 Keep-Alive mechanism

#### 5.5.1.3 Introduce a cyclical Alive Check

As defined in ISO 13400-2 (refer to [3] or [4]), alive check requests can be sent and alive check responses are expected. The requests can be sent cyclically from client and server or on establishment of a new connection on server side. This is not implemented natively in DoIPInt but can be modelled by the user when using OEM specific payload types (refer to chapter 5.4.3). One disadvantage is that when multiple logical addresses are available, it is not clear which one shall be used in the related message field. A workaround to this would be to use a customized alive check response with an own payload type since it is anyway not supported natively. Like this, it can be prevented to use the logical address in the alive check response message. Figure 5-15 illustrates this mechanism.

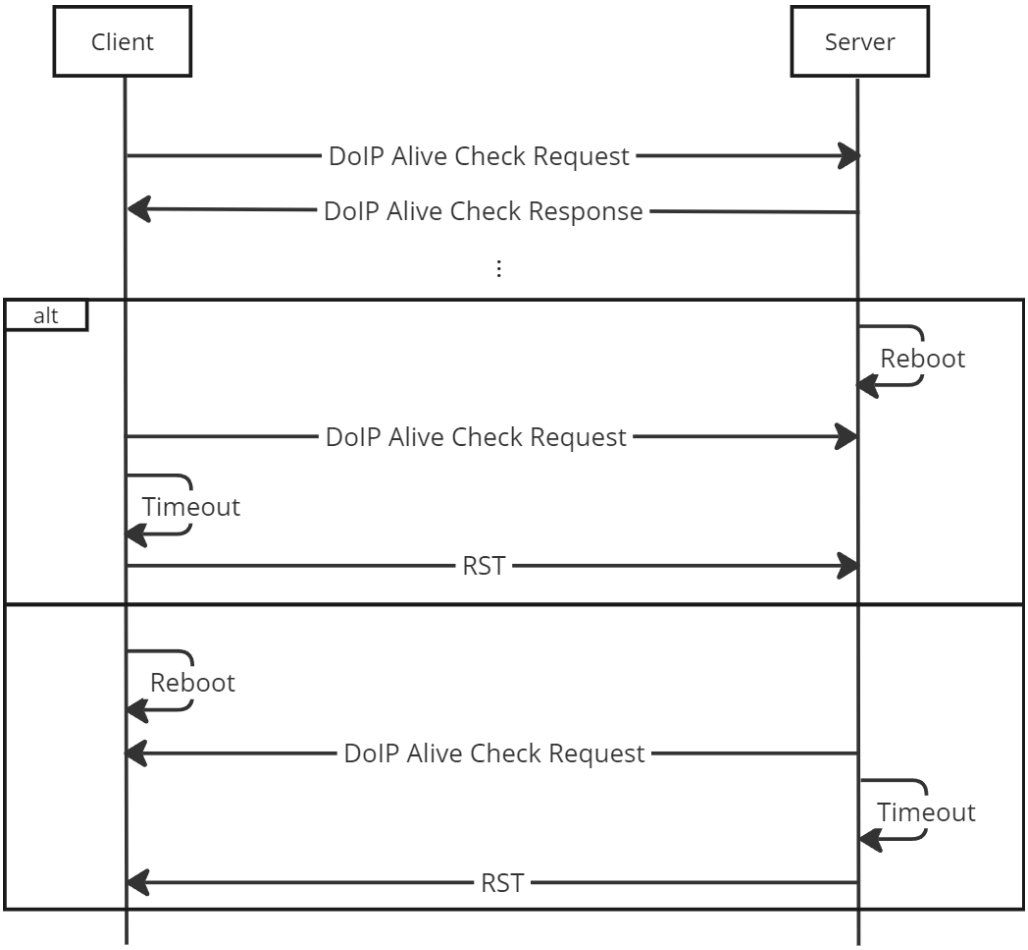


Figure 5-15 Cyclical Alive Checks

# 6 Glossary and Abbreviations

## 6.1 Glossary

| Term | Description |
|---|---|
| BswM | AUTOSAR module Basic Software Mode Manager |
| PduR | AUTOSAR module PDU Router |
| SchM | AUTOSAR module Schedule Manager |
| SoAd | AUTOSAR module Socket Adaptor |
| TcpIp | AUTOSAR module TcpIp |

Table 6-1 Glossary

## 6.2 Abbreviations

| Abbreviation | Description |
|---|---|
| API | Application Programming Interface |
| AUTOSAR | Automotive Open System Architecture |
| BSW | Basis Software |
| DCM | Diagnostic Communication Manager |
| DET | Development Error Tracer |
| ECU | Electronic Control Unit |
| MICROSAR | Microcontroller Open System Architecture (the Vector AUTOSAR solution) |
| OS | Operating System |
| TCP | Transmission Control Protocol |
| UUDT | Unacknowledged Unsegmented Data Transfer |

Table 6-2 Abbreviations

# 7 Contact

Visit our website for more information on

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

www.vector.com