

MICROSAR Classic PDU Router

Technical Reference

DaVinci Configurator

Version 10.00.01

Authors	visehs, visms, visswa, visfrm, visbbk, vishho, vismij, visaol
---------	--

Status	Released
--------	----------

Document Information

History

Author	Date	Version	Remarks
	2012-12-20	1.00.00	Initial version based on PduR Technical Reference
visehs	2012-07-12	2.00.00	Adapted to AUTOSAR 4.0.3
visehs	2012-10-15	2.01.00	TP Gateway IF Gateway
visms	2012-11-21	2.02.00	AR4-285: Support PduRRoutingPathGroups
visehs	2013-02-07	2.02.01	Adapted Tp- API description
visehs	2013-02-15	2.02.02	Added some ASR deviations ESCAN00064126
visehs	2013-03-19	2.03.00	ESCAN00064364 AR4-325: Post-Build Loadable Added Cancel- Receive/ Transmit Support
visehs	2014-04-15	2.04.00	Added TP routing with variable addresses (MetaData Handling) Added Threshold "0" support
visehs	2014-04-15	2.04.01	Support the StartOfReception API (with the PduInfoType), TxConfirmation and RxIndication according ASR4.1.2
visehs	2014-09-01	2.05.00	Added SecOC to the Interface Overview Extended Tp Gateway Routing behavior description Updated Configuration Variant
visswa	2015-02-23	2.06.00	FEAT-1057: Added documentation about configuration of range routing paths and functional requests gateway
visswa	2015-05-11	2.06.01	FEAT-1057: Improvements of documentation
visfrm	2015-07-30	2.07.00	FEAT-109: Added documentation for PduR switching feature and N:1 routing paths
visfrm	2016-01-16	2.08.00	FEAT-1485: Added documentation for 1:N and N:1 transport protocol routing paths
visms	2016-02-25	2.08.00	FEAT-1631: Trigger Transmit API with SduLength In/Out according to ASR4.2.2
visehs	2016-03-17	2.08.00	added limitation: - The Polling Mode cannot be used for N:1 routings. - Cancel Transmit for N:1 routing paths is only supported if a Tx Confirmation is enabled. - Removed limitation: N:1 interface routing paths support only for lower layer CanIf.
visfrm	2016-04-01	2.08.01	Removed empty chapters
visehs, visfrm	2016-08-10	3.00.00	Shared/Dedicated Buffer support Memory mapping extension
visswa	2016-11-24	3.00.00	Smart Learning (Switching)

visfrm	2017-06-22	3.00.01	ESCAN00095254: Missing DET error PDUR_E_PDU_INSTANCES_LOST description in case of N:1 communication interface routings with upper layer
visfrm	2017-06-23	3.00.01	STORYC-1629: N:1 routing path support for IpduM Container feature
visfrm	2017-09-21	3.01.00	STORYC-1972: Support 1:N Tx IF API Forwarding
visbbk	2017-11-07	3.02.00	STORYC-2031: Support buffered IF API Forwarding routing paths
visfrm	2018-01-30	3.03.00	STORYC-3416: PduR: Enable assignment of buffers smaller than the associated PDU sizes to the buffer pool referenced by a routing path STORYC-3435: PduR: Enable routing paths with different connected PDU sizes in gateway routings
vishho	2018-03-07	3.04.00	Maintenance (STORYC-3934)
vishho	2018-03-14	4.00.00	STORYC-3241: Implement Multicore Support
visfrm	2018-07-04	4.01.00	STORYC-4890: Release PduRBswModules on different Cores
visfrm	2019-02-25	5.00.00	STORYC-7435: Release [GW] Message Buffering in MICROSAR Classic
visfrm	2019-04-17	5.01.00	STORYC-7752: On-the-fly routing of (J1939) TP messages
visfrm	2019-08-12	5.02.00	ESCAN00103997: Old term 'TpBuffer' is still used instead of 'TxBuffer'
visfrm	2019-10-17	6.00.00	MSR4-117: PduR Support for Mixed ASIL Systems
visfrm	2019-11-21	6.01.00	COM-1247: Improve documentation for Mixed ASIL MultiPartition UseCase
vismij	2020-02-27	6.02.00	COM-1409: PduR_Prelit documentation
visfrm	2020-07-07	7.00.00	COM-1300: Provide MC Queue Configuration Per Queue
visfrm	2020-09-29	7.01.00	COM-1898: Document MainFunction in PduR TechRef
visfrm	2020-10-15	8.00.00	COM-1850: Support Partition mapping via the GlobalPdu
visfrm	2020-12-07	8.01.00	COM-2158: Support Routing Paths from an Upper Layer to another Upper Layer COM-2001: Remove the AUTHOR IDENTITY
visfrm	2021-02-02	8.02.00	COM-2258: PduR: Support Dedicated Partition Reference Parameter COM-1973: Update Doc_TechRef with the new template
visfrm	2021-07-13	8.02.01	ESCAN00109725: Incorrect function signature documentation for Appl_(Get TryToGet Release)Spinlock()
viessg	2021-08-30	8.03.00	ESCAN00110076: Missing description for Memory sections mapping for Multi-Partition use case
vislan	2022-03-01	9.00.00	Product name updated to MICROSAR Classic.
visfrm	2022-07-27	9.00.00	COM-3858: Remove PduR switching functionality
visaol	2023-01-31	10.00.00	COM-4291: Improve documentation
oalka	2023-11-27	10.00.01	ESCAN00109962: Missing documentation for PduR memory section 'VAR_ZERO_INIT_UNSPECIFIED'

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	Specification of PDU Router	4.2.2
[2]	AUTOSAR	Specification of Development Error Tracer	3.2.0
[3]	AUTOSAR	List of Basic Software Modules	1.6.0
[4]	AUTOSAR	Specification of a Transport Layer for SAE J1939	1.5.0
[5]	Vector	TechnicalReference CanIf	see delivery
[6]	Vector	TechnicalReference <CAN Driver>	see delivery
[7]	Vector	TechnicalReference CanTp	see delivery
[8]	Vector	UserManual Multi-Core	see delivery

This technical reference describes the general use of the PduR basis software module.



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

1	Introduction.....	9
1.1	Architecture Overview	10
2	Functional Description.....	11
2.1	Features	11
2.1.1	Deviations	11
2.1.2	Additions/ Extensions.....	12
2.1.3	Limitations.....	12
2.2	Initialization	12
2.3	States	12
2.4	Error Handling.....	13
2.4.1	Development Error Reporting.....	13
2.5	Communication Interface Routing	14
2.5.1	Data Provision.....	14
2.5.1.1	Direct data provision	14
2.5.1.2	Trigger transmit data provision.....	15
2.5.2	Queued IF routing	15
2.5.2.1	FIFO Queue.....	15
2.5.2.2	Single Buffer Queue.....	15
2.5.3	Timing aspects	15
2.5.4	Dynamic DLC Routing.....	15
2.5.5	Transport protocol low level routing.....	16
2.5.6	Queue overflow notification callback	16
2.5.7	N:1 Routing Paths with Upper Layer and Tx confirmation	18
2.6	Transport Protocol Routing	18
2.6.1	Multi-Routing.....	18
2.6.2	TP Threshold	18
2.6.2.1	Restrictions.....	19
2.6.2.2	Threshold “0”	20
2.6.3	Queued TP routing	20
2.6.3.1	Supported Queues.....	20
2.6.3.2	TxBuffer Handling	20
2.6.4	Error Handling.....	20
2.6.5	Meta Data Handling	21
2.7	Queues	22
2.7.1	Single Buffer Queue.....	22
2.7.2	Communication Interface Queue.....	23
2.7.3	Shared Buffer Queue	23
2.7.3.1	Tx Buffer Assignment.....	23

2.7.3.1.1	Explicit Tx Buffer assignment	23
2.7.3.1.2	Implicit Tx Buffer assignment	24
2.7.3.2	Tx Buffer Length Configuration	24
2.7.3.2.1	Communication Interface	24
2.7.3.2.2	Transport Protocol	25
2.7.3.3	Tx Buffer Selection Algorithm	25
2.7.4	Queue Depth	25
2.8	Cross Partition Routing	26
2.8.1	General configuration	26
2.8.2	Multicore Queue	26
2.8.3	Deferred Event Cache	27
2.8.4	Spinlocks	28
2.8.4.1	SpinLock Retry Counter	29
2.8.5	Restrictions	29
2.9	PduR in mixed ASIL Systems	30
2.9.1	Routing paths with safety related availability requirements	30
2.9.2	Routing paths without safety requirements	30
3	Integration	31
3.1	Scope of Delivery	31
3.1.1	Static Files	31
3.1.2	Dynamic Files	33
3.2	Critical Sections	34
3.3	Memory Sections	34
3.3.1	Memory sections for PduRTxBuffer	34
3.3.2	Memory sections for Multi-partition use case	34
3.4	Type Definitions	35
4	API Description	36
4.1	Services provided by PDUR	36
4.1.1	PduR_PreInit	36
4.1.2	PduR_Init	37
4.1.3	PduR_InitMemory	37
4.2	Services	38
4.2.1	PduR_GetVersionInfo	38
4.2.2	PduR_GetConfigurationId	38
4.2.3	PduR_EnableRouting	39
4.2.4	PduR_DisableRouting	39
4.2.5	PduR_MainFunction	40
4.3	Communication Interface	41
4.3.1	PduR_<GenericUp>Transmit	41

4.3.2	PduR_<GenericLo>RxIndication	42
4.3.3	PduR_<GenericLo>TriggerTransmit.....	43
4.3.4	PduR_<GenericLo>TxConfirmation	44
4.4	Transport Protocol	45
4.4.1	PduR_<GenericUpTp>ChangeParameter.....	45
4.4.2	PduR_<GenericUpTp>CancelReceive	46
4.4.3	PduR_<GenericUpTp>CancelTransmit	47
4.4.4	PduR_<GenericLoTp>StartOfReception	48
4.4.5	PduR_<GenericLoTp>CopyRxData	49
4.4.6	PduR_<GenericLoTp>CopyTxData.....	50
4.4.7	PduR_<GenericLo>TpTxConfirmation	51
4.4.8	PduR_<GenericLo>TpRxIndication.....	52
4.4.9	PduR_<GenericUpTp>Transmit	53
4.5	Service Ports	54
4.5.1	Complex Device Driver Interaction	54
5	Configuration	55
5.1	Use Case Configuration: Communication interface range gateway	55
5.1.1	Step-by-step configuration	56
5.1.2	Optional configuration variants / options.....	61
5.2	Use Case Configuration: Functional requests gateway routing	63
5.2.1	Step-by-step configuration	64
5.3	Use Case Configuration: N:1 routing path.....	71
6	Glossary and Abbreviations	74
6.1	Glossary	74
6.2	Abbreviations	75
7	Contact	77

Illustrations

Figure 1-1	Interfaces to adjacent modules of the PDUR	10
Figure 2-1	Overflow notification callback configuration	17
Figure 2-2	Configured Tp Buffer with possible Threshold ranges	19
Figure 5-1	Meta data routing with CanIf	55
Figure 5-2	CanIf / PduR range routing example overview	57
Figure 5-3	Example functional requests gateway network architecture	63
Figure 5-4	Functional request gateway architecture	63
Figure 5-5	example N:1 routing path configuration	72
Figure 5-6	EcuC configuration of (mixed) N:1 / 1:N routing paths	73

Tables

Table 2-1	Supported AUTOSAR standard conform features	11
Table 2-2	Not supported AUTOSAR standard conform features	12
Table 2-3	Features provided beyond the AUTOSAR standard	12
Table 2-4	Errors reported to DET	14
Table 2-5	How to get the associated gateway routing path	17
Table 3-1	Static files	32
Table 3-2	Generated files	33
Table 3-3	Memory Mapping in multi-partition use case	34
Table 3-4	Type definitions	35
Table 4-1	PduR_Init	37
Table 4-2	PduR_InitMemory	37
Table 4-3	PduR_GetVersionInfo	38
Table 4-4	PduR_GetConfigurationId	38
Table 4-5	PduR_EnableRouting	39
Table 4-6	PduR_DisableRouting	39
Table 4-7	PduR_MainFunction	40
Table 4-8	PduR_<GenericUp>Transmit	41
Table 4-9	PduR_<GenericLo>RxIndication	42
Table 4-10	PduR_<GenericLo>TriggerTransmit	43
Table 4-11	PduR_<GenericLo>TxConfirmation	44
Table 4-12	PduR_<GenericUpTp>ChangeParameter	45
Table 4-13	PduR_<GenericUpTp>CancelReceive	46
Table 4-14	PduR_<GenericUpTp>CancelTransmit	47
Table 4-15	PduR_<GenericLoTp>StartOfReception	48
Table 4-16	PduR_<GenericLoTp>CopyRxData	49
Table 4-17	PduR_<GenericLoTp>CopyTxData	50
Table 4-18	PduR_<GenericLo>TpTxConfirmation	51
Table 4-19	PduR_<GenericLo>TpRxIndication	52
Table 4-20	PduR_<GenericUpTp>Transmit	53
Table 5-1	Example range routings	57
Table 5-2	Example functional diagnostic request routing	65
Table 6-1	Glossary	75
Table 6-2	Abbreviations	76

1 Introduction

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

Supported Configuration Variants:	PRE-COMPILE [SELECTABLE] POST-BUILD-LOADABLE [SELECTABLE]	
Vendor ID:	PDUR_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	PDUR_MODULE_ID	51 decimal (according to ref. [3])

The main task of the PDU Router module is to abstract from the type of bus access (Interface layer and TP layer) and from the bus type itself.

Since the PDU Router module has to route Rx and Tx PDUs to and from the upper- and lower- layers and any software component uses its own handle space, multiple routing tables are required. The PDU Router uses the input handle as an index to the related routing table.

1.1 Architecture Overview

Figure 1-1 shows the interfaces to adjacent modules of the PDUR. These interfaces are described in chapter 4.

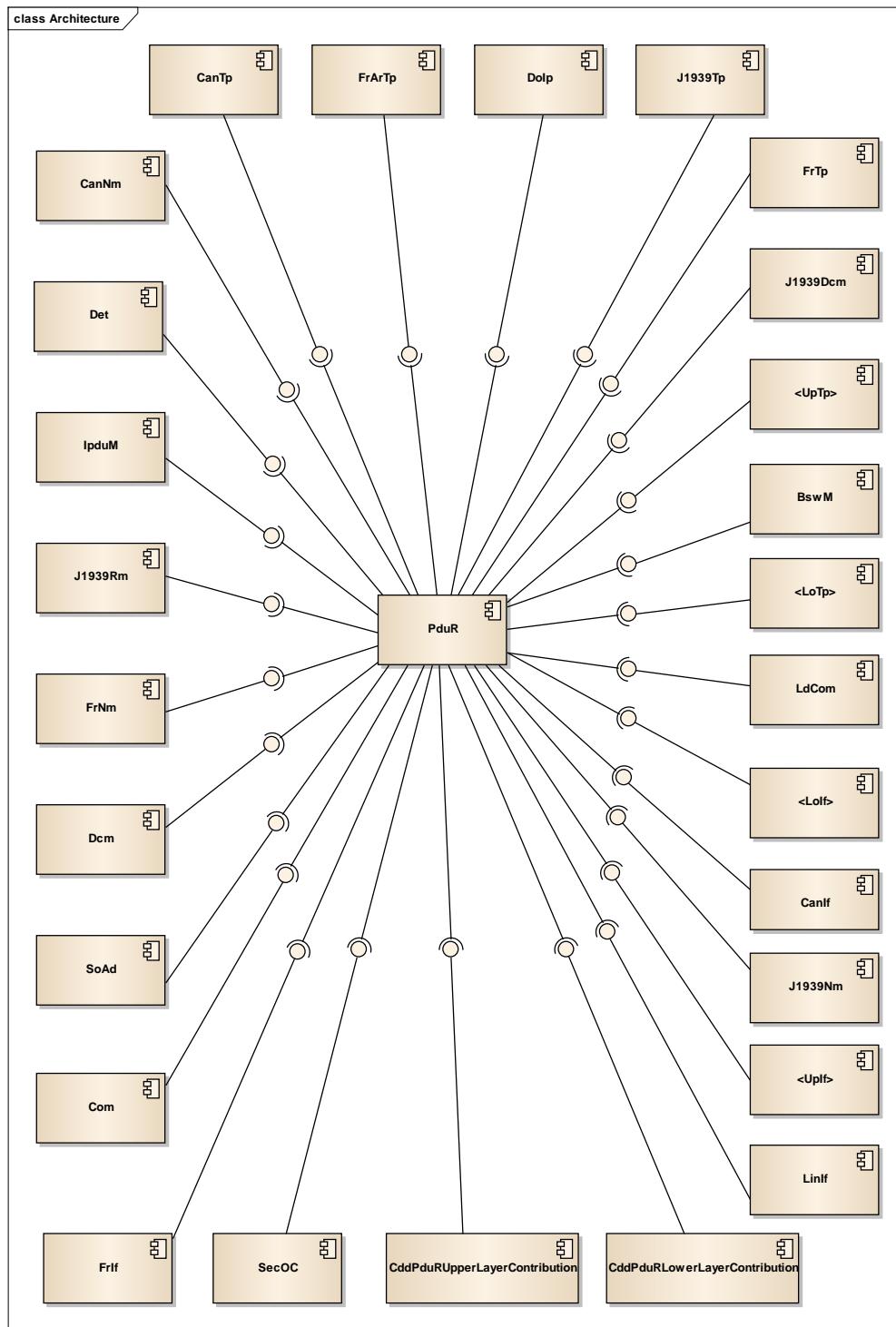


Figure 1-1 Interfaces to adjacent modules of the PDUR

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 the PDUR are listed in chapter 4 and are defined in [1].

2 Functional Description

2.1 Features

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

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

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

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

- ▶ Table 2-3 Features provided beyond the AUTOSAR standard

The following features specified in [1] are supported:

Supported AUTOSAR Standard Conform Features
Pre compile and Post build time configuration variant
I-PDU transmission and reception
Cancel-Receive/Transmit support
Change Parameter support
1:1 routing between upper- and lower-layer communication interface modules
1:1 routing between upper- and lower-layer transport protocol modules
1:1 Interface Gateway Routing
1:N Interface Gateway Routing
1:1 Transport protocol Gateway Routing
1:N Transport protocol Gateway Routing (single and multiframe Tp messages)
Complex device driver (CDD) support
Routing Path Groups
Debugging support (optional feature)
1:N fan-out from the same upper layer PDU (If)

Table 2-1 Supported AUTOSAR standard conform features

2.1.1 Deviations

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

Category	Description
Config	Link time configuration
Functional	1:N fan-out from the same upper layer PDU (Tp)
Functional	Zero cost operation

Category	Description
API	PduR_<User:LoTp>CopyTxData: The retry parameter is not supported.
API	PduR_(Enable Disable)Routing: If the routing path group id does not exist, then the API will report a DET error instead of returning with no action.

Table 2-2 Not supported AUTOSAR standard conform features

2.1.2 Additions/ Extensions

The following features are provided beyond the AUTOSAR standard:

Features Provided Beyond The AUTOSAR Standard
N:1 Interface routing path capability (destinations with polling behaviour are not supported)
1:N and N:1 transport protocol single- and multiframe routing path capability
Choose between different Queues/Queue Implementations
Routing between upper layers

Table 2-3 Features provided beyond the AUTOSAR standard

2.1.3 Limitations

Since 8-bit micro controllers are out of scope in AUTOSAR, PDUR has been optimized for the usage on 16- and 32-bit controllers. Therefore the target system must be able to provide atomic read and write accesses to 16-bit variables.

2.2 Initialization

Before the PduR can be used it has to be initialized by PduR_PrelInit() and PduR_Init().

PduR_PrelInit has to be called only once by the EcuMDriverInitListOne before the start of the OS.

Afterwards PduR_Init has to be called once per Partition/OsApplication where the PduR is running.

2.3 States

The PduR is initially uninitialized. Basic RAM arrays are initialized with the call of PduR_InitMemory or with the startup code of your ECU.

PduR_PrelInit initializes all shared memory used by the PduR. Afterwards it is in the Preinitialized state.

PduR_Init initializes all Partition/OsApplication specific memory. It has to be called once on every Partition/OsApplication where the PduR is running. Afterwards the state of the PduR on this Partition/OsApplication is Initialized.

2.4 Error Handling

2.4.1 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 `PDUR_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 PDUR ID is 51.

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

Error Code	Description
PDUR_E_INVALID_REQUEST	An API service has been used without a call of PduR_Init() or PduR_Init() was called while the PDU Router is initialized. If your system does not use a start-up code to initialize global variables, PduR_InitMemory() can be called before PduR_Init() to avoid this problem.
PDUR_E_PDU_ID_INVALID	An invalid PDU identifier was used as parameter for a PDU Router API call.
PDUR_E_ROUTING_PATH_GROUT_ID_INVALID	If the routing table is invalid that is given to the PduR_EnableRouting or PduR_DisableRouting functions
PDUR_E_INIT_FAILED	PDU Router initialization failed
PDUR_E_PDU_INSTANCES_LOST	Loss of a PDU instance (FIFO flushed because of an overrun)
PDUR_E_UNEXPECTED_CALL	The indicated API was called although the current PDU Router configuration and internal state does not expect a call to this API
PDUR_E_FATAL	Fatal error
PDUR_E_MCQ_QUEUE_OVERFLOW	The multicore queue has overflowed.
PDUR_E_INVALID_APPLICATION_ID	The PduR Mainfunction has been called in an invalid Partition/OsApplication context.
PDUR_E_SPINLOCKTIMEOUT	The PduR ran into a timeout while trying to acquire a spinlock.
PDUR_E_IDXOUTOFCOMM	A PduR variable was read which had an invalid value.
PDUR_E_OSAPP2MEMMAPPIN	The OsApplication ID could not be mapped to a memory section.
PDUR_E_PARAM_POINTER	A pointer parameter is invalid.
PDUR_E_NO_PREINIT	The PduR has not be pre-initialized.
PDUR_E_ALREADY_INITIALIZED	The PduR was already initialized.
PDUR_E_QAL_UNSUPPORTED_API	A QAL API is not supported for this destination Pdu.
PDUR_E_INVALID_PARTITION_CONTEXT	An API was called in the wrong partition context.

Table 2-4 Errors reported to DET

2.5 Communication Interface Routing

2.5.1 Data Provision

2.5.1.1 Direct data provision

If the data provision is set to ‘Direct’ the Pdu will be copied to the destination in the Transmit function. The Transmit function of the destination module is called by the PduR.

2.5.1.2 Trigger transmit data provision

If the data provision is set to ‘Trigger Transmit’ the Pdu will be copied to the destination in the TriggerTransmit function. The TriggerTransmit function of the PduR is called by the destination module. This is useful if the destination module always wants to fetch the latest data available (single buffer configuration) or it has specific timing requirements when it needs to provide the data.

2.5.2 Queued IF routing

Queueing is enabled by referring a PduRQueue container at the PduRDestPdu.

The following queues are supported by communication interface routing paths:

- ▶ Communication Interface Queue (page 23): FIFO behavior
- ▶ Shared Buffer Queue (page 23): FIFO behavior
- ▶ Single Buffer Queue (page 22): Last-is-best behavior

2.5.2.1 FIFO Queue

A FIFO is used if loss of I-PDU instances is critical. One FIFO queue only cares for a FIFO based sorting of the instances of its own queued I-PDUs. The queue can be configured for each destination independently.

The queue will be flushed in the following situations:

- ▶ A queue overflow (the newly received I-PDU will be routed)
- ▶ Transmission to the destination is not successful
- ▶ A corresponding PduRDestPdu is disabled by routing path groups
- ▶ A suitable TxBuffer could not be allocated (Shared Buffer Queue only)

2.5.2.2 Single Buffer Queue

This queue has a ‘Last-is-best’ behavior. The single buffer will always be overwritten with the latest received Pdu. This way only the latest value is provided to the destination.

For this behavior the data-provision of all PduRDestPdus using this queue must be set to ‘trigger transmit’.

The data of the buffer can then be read at any time via the TriggerTransmit API. The single Buffer is initialized with the default values configured at the PduRDestPdu. This way the TriggerTransmit API can also be called even if no Pdu has yet been received.

2.5.3 Timing aspects

The PDU Router does not provide a mechanism to implement a rate conversion (e.g. change the cycle time from the source to the destination channel). A rate conversion can be implemented by signal routing paths using the COM signal gateway.

2.5.4 Dynamic DLC Routing

The length of a received I-PDU can be changed dynamically at runtime. It may vary between 0 and the configured PDU length.

Dynamic length I-PDUs can be used to save memory resources. This only applies if a Shared Buffer Queue is used:

If the routing path receives smaller I-PDUs most of the time, you can assign smaller buffers to the queue. The smaller buffers will be used if they are large enough for the actual received message. An example use case is the routing of classical CAN messages via a CAN-FD routing path. In this case an 8 byte I-PDU does not occupy a 64 byte buffer.

**Caution**

A Pdu with a DLC larger than the configured Pdu length will be truncated to the global Pdu length of this routing path or will be silently discarded depending on the setting of parameter PduRPduLengthHandlingStrategy.

2.5.5 Transport protocol low level routing

If the TP segments (N-PDUs) on the source and the destination network are identical, it is possible to route TP I-PDUs using the interface layer gateway ("low-level" routing). If low-level routing is used, the (former) N-PDU is no longer accessible to the TP layer and therefore seen as an I-PDU by the PDU Router module.

The advantage of low-level routing is that it is executed in the context of the interface layer RxIndication and therefore introduces a minimal routing latency.

Low-level routing has, however, several drawbacks which might cause that a high-level TP routing is more adequate:

- ▶ TP protocol conversion is not possible as the frame-layout and the flow-control handling must be the same on the source and the destination network.
- ▶ No forwarding of routed TP I-PDUs to the local DCM is possible as it may be required for functional requests.
- ▶ Eventual loss of TP parameters, such as the STmin timing and the block size, due to bursts on the destination bus. Bursts are a result of queued I-PDUs which were transmitted in the TxConfirmation of the previous I-PDU.
- ▶ A buffer overrun in the FiFo queue causes the queue to be flushed and therefore corrupts the TP communication. The TP layer gateway can avoid buffer overflows if the receiving TP connection supports a dynamic block size adaptation.

2.5.6 Queue overflow notification callback

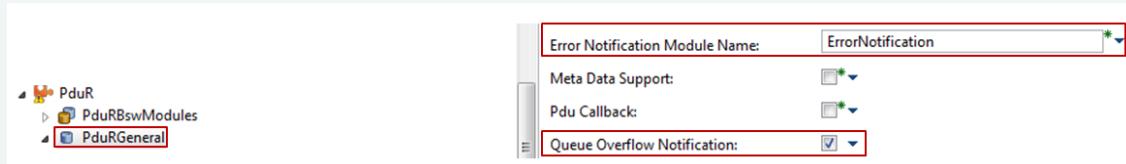
The PDU Router supports a Queue overflow notification callback. In case of a communication interface routing with unfavorable FIFO configuration or if the destination bus is not available (e.g. bus off) a buffer overflow can occur. In this case the FIFO is flushed.

Additionally, a callback could be configured to capture this event and perform error handling. See Figure 2-1.



Enable Feature overflow notification callback

- > PduR_QueueOverflowNotification activates / deactivates a PduR communication interface TxBuffer overflow notification.
- > Set an individual error notification module name by using PduR_ErrorNotificationModuleName parameter. This parameter is optional and can be left empty. ‘Error Notification’ is used as default value.



If the feature is enabled two additional source files are generated to the Gendata folder. A <ErrorNotification>_Cbk.h file and a _<ErrorNotification>_Cbk.c template file. The template contains the error notification function which must be implemented.

- > Implement the error notification function and remove the template underscore.

Figure 2-1 Overflow notification callback configuration

During runtime the error notification is called by the PDU Router in case of a FIFO overflow. The lower layer interface handle is passed to the notification callback.



How to get the associated gateway routing path

- > Open the Find view and enter the destination PDU name with the associated handle. Syntax: value == destination PDU name
- > Right click on the PDU Router destination container in the result view and open the reference using the “Show referenced item in” dialog.

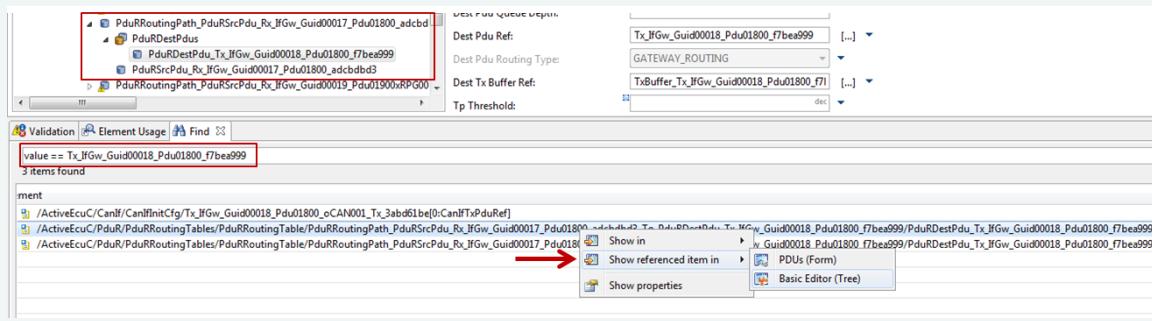


Table 2-5 How to get the associated gateway routing path

**Caution**

The error notification function is called in the interrupt context! Please keep the error handling short.

2.5.7 N:1 Routing Paths with Upper Layer and Tx confirmation

**Caution**

If a N:1 communication interface routing path includes an upper layer source with enabled Tx confirmation, the destination will be locked when a Pdu is routed from the upper layer to the destination until the upper layer receives the corresponding Tx confirmation. While this lock is active no other Pdu can be routed on this routing path (neither gateway nor API forwarding routings). All corresponding transmit requests will not be executed and a DET error will be reported if enabled.

This behavior can be avoided if no Tx Confirmation is configured for the upper layer source.

2.6 Transport Protocol Routing

The PduR allows high-level routing of TP I-PDUs.

In order to reduce runtime and memory consumption, the PduR supports routing on-the-fly (for 1:1, 1:N and N:1 single and multiframe routing paths). Depending on the “Threshold” configuration of each routing path, the gateway starts with the transmission on the destination network before the reception has been completed on the source network.

2.6.1 Multi-Routing

The PduR supports N:1 and 1:N routing paths for both single- and multiframe. Each of these routing paths uses a single queue at runtime.

A 1:N gateway routing path can be combined with a Rx API forwarding routing path to the upper layer. The queuing of the forwarding is optional. If no queuing is used for the forwarding it behaves like a normal 1:1 forwarding routing path. The returned values of the upper layer and the internal PduR Rx APIs are merged:

- ▶ The smaller of the two bufferSize is returned to the caller.
- ▶ If one API fails, the negative returned value is returned to the caller.

For N:1 routing paths the parallel reception from different sources is supported, but the PDUs will be serialized and put to the queue. From there they will be transmitted one after the other.

2.6.2 TP Threshold

The Threshold value is used to...

- ▶ ... define the fill level of the buffer which triggers the transmission on the destination bus.

- ▶ ... exclude buffers from the selection during runtime. This could be helpful to ensure that small buffers which are configured for single frames are not taken into account for long multi messages. If the Threshold is larger than the buffer size it is ensured that the PduR will not use this buffer to perform the routing.



Note

Small buffers can also be excluded from the selection at runtime by assigning all suitable buffers to the respective routing paths manually. Only these assigned buffers will be used at runtime. See chapter 2.7.3.1 for details.

2.6.2.1 Restrictions

The setting of the TP Threshold is restricted for CanTp, LinTp and FrArTp routing paths. Threshold values in the shaded sections of Figure 2-2 are not allowed to be configured for these kind of gateway routings.

Each PduR_<LoTp>_CopyRxData() call requires that a complete consecutive frame (CF) will be copied to the buffer. If not enough buffer is available the Tp tries again to get more buffer. But the PduR still cannot dequeue data because the threshold is not reached. In this situation there will never be enough buffer space and the routing will be aborted with a timeout.

The DaVinci Configurator validates this and provides appropriate Solving-Actions to set a suitable Threshold. The recommended values are adapted to the boundaries.

If just the largest buffer of Figure 2-2 should be used for a routing the Threshold must be set to Section 2. If the routing should have both buffer options the Threshold level must be set somewhere in Section 1.

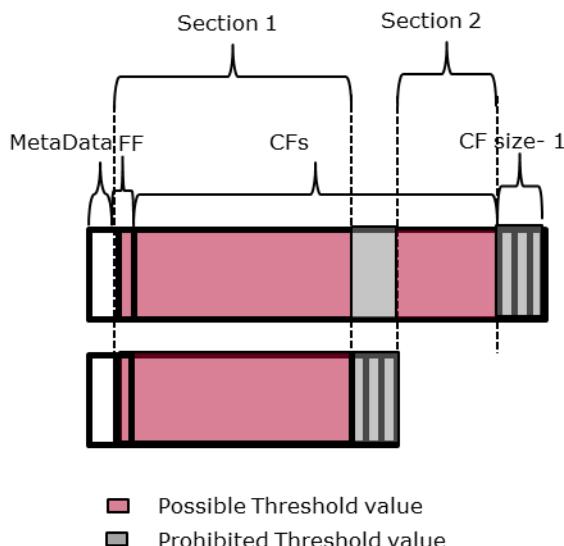


Figure 2-2 Configured Tp Buffer with possible Threshold ranges

2.6.2.2 Threshold “0”

The PduR module supports a Threshold of “0”. This means that the transmission will be triggered immediately. The first frame of some Transport Protocol modules (J1939Tp and FrTp) does not contain data but it is required that the transmission will be triggered nevertheless. For further details refer for example the J1939Tp specification [4].

2.6.3 Queued TP routing

2.6.3.1 Supported Queues

A transport protocol routing path only supports the Shared Buffer Queue. (see page 23)

This queue supports FIFO behavior. This means that the first started source TP connection is transmitted to the destination first.

Every PduRDestPdu can be configured to use a queue to buffer the TP I-PDUs by referring a PduRQueue container.

TP Queues are supported for the following routing paths:

- ▶ 1:1 routing path
- ▶ 1:N routing path (the same Queue must be referred by all destinations)
- ▶ N:1 routing path (same Queue must be referred by all DestPdus which refer the one common destination global PDU)

Multiple I-PDUs from different source connections can be received at once on N:1 routing paths, as long as the TP Queue is not full. For routing paths with only one possible source Tp connection (1:1, 1:N), only one I-PDU can be received at once. If the transmission on the destination side is delayed, multiple I-PDU instances which were received on the source Tp connection are stored in the queue.

2.6.3.2 TxBuffer Handling

Diagnostic communication usually does not take place within all ECUs at the same time. Therefore a TP routing path typically has no explicitly assigned buffer. In order to reduce the amount of RAM required for queues, the Tx buffers are assigned dynamically to active TP routing paths and can be reused in different routing paths automatically.

If a buffer is assigned to a routing path during runtime, this buffer is exclusively reserved for this routing. If all available buffers are occupied, no further routing is possible and the PduR signalizes the state “BUFREQ_E_NOT_OK” towards the TP module. It will then create an appropriate flow-control frame depending on its capabilities.

2.6.4 Error Handling

If one of the source or destination TP components that are involved in a TP data transfer stops its transmission or reception due to an error (e.g. a timeout has occurred), the corresponding TP-routing relation and buffer will be released. The next buffer request on the not yet aborted Tp connection will return "BUFREQ_E_NOT_OK" and will release the Tp connection.

**Info**

There is no AUTOSAR mechanism which notifies the other TP component side of an error during the reception or transmission.

2.6.5 Meta Data Handling

Since ASR 4.1.2 the StartOfReception() API was extended by the “PduInfoPtr”. This parameter can be used to transmit meta data.

In case of a gateway routing the payload provided in the “PduInfoPtr” will be ignored by the PduR. Only the meta data is buffered and transmitted via the <LL>_Transmit function. In case of forwarding to an upper layer module (e.g. DCM or CDD) the payload and meta data will be forwarded and the upper layer must extract the meta data according the configured meta data length.

**Caution**

The length of the “PduInfoPtr” does not contain the “MetaDataLength” information. The length parameter represents the I-PDU total length. Each module must copy the MetaData from the “PduInfoPtr” according the configured “MetaDataLength”. Do **not use** the length of the “PduInfoPtr” to copy “MetaData” from the “PduInfoPtr”.

**Caution**

The lower layer module must provide the payload in the CopyRxData() function. It can not provide it in the StartOfReception call.

2.7 Queues

**EcuC structural changes with PduR version 15.03**

With this PduR version a new model structure was introduced. Queues can now be created explicitly. This is done via the PduRQueue container. For each queue a different type/implementation can be chosen.

In old versions a queue was configured by these parameters:

- ▶ PduRRoutingPath/PduRDestPdu/PduRDestPduQueueDepth
- ▶ PduRRoutingPath/PduRDestPdu/PduRTpThreshold (TP only)
- ▶ PduRRoutingPath/PduRDestPdu/PduRDestTxBufferRef

These parameters were moved to:

- ▶ PduRQueue/PduRSharedBufferQueue/PduRQueueDepth
- ▶ PduRQueue/PduRSharedBufferQueue/PduRTpThreshold
- ▶ PduRQueue/PduRSharedBufferQueue/PduRExplicitTxBufferRef

Additionally, new queue types were introduced:

- ▶ PduRQueue/PduRCommunicationInterfaceQueue
- ▶ PduRQueue/PduRSingleBuffer

To use a queue in a routing path it must be referred by the routing path:

- ▶ PduRRoutingPath/PduRDestPdu/PduRQueueRef

2.7.1 Single Buffer Queue

This queue can only be used by communication interface routing paths with a data provision of trigger transmit.

A single buffer queue has no additional parameters. It always has a queue depth of one. The queue will always be overwritten by the new Pdu put into the queue. This way a last-is-best behavior is implemented.

2.7.2 Communication Interface Queue

This queue can only be used by communication interface routing paths. It has a FIFO (first in, first out) behavior. The only parameter is the queue depth.



Caution

It is recommended to use this queue for all queued communication interface routing paths. This queue is optimized for these simple routing paths and therefore results in less memory overhead compared to a Shared Buffer Queue.

2.7.3 Shared Buffer Queue

This queue can be used by communication interface and transport protocol routing paths. It has a FIFO (first in, first out) behavior.



Note

This queue corresponds to the same queue used in older PduR version where there was no explicit PduRQueue in the model.

This queue has the most flexibility by using several configuration parameters:

- ▶ PduRQueueDepth
- ▶ PduRTpThreshold (only used for transport protocol routing paths)
- ▶ PduRExplicitTxBufferRef: PduRTxBuffer which is configured by the user to be used by this queue
- ▶ PduRImplicitTxBufferRef: PduRTxBuffer which will implicitly be used if the user has not configured PduRExplicitTxBufferRef. This is calculated automatically and shall not be changed manually.
- ▶ PduRLockRef: This parameter is only relevant for multicore configurations. It is calculated automatically in this case and shall not be change manually.

2.7.3.1 Tx Buffer Assignment

2.7.3.1.1 Explicit Tx Buffer assignment

If a PduRTxBuffer is assigned to a shared buffer queue manually by the user (via PduRExplicitTxBufferRef), it is called an explicit Tx Buffer assignment. The queue will only use these explicitly assigned buffers. A validator ensures that you assign a suitable number of buffers.



Dedicated Tx Buffer

If a PduRTxBuffer is referenced by only **one** queue, it is called a dedicated Tx Buffer. The buffer can only be used by this queue.

Dedicated Tx Buffer accelerate the buffer search algorithm.

Dedicated Tx Buffer can be used to ensure the availability of suitable Tx.



Expert Knowledge

Routing of a functional request is a typical use case for a dedicated buffer. For a short diagnostic request, searching for a buffer shall be avoided. If a dedicated buffer is assigned to the `PduRDestPdu`, this buffer will be used during runtime.



Note

Dedicated Tx Buffers raise the RAM consumption. Every Tx Buffer allocates its needed memory in RAM. This memory will not be shared with any other routing path.



Shared Tx Buffer

A PduRTxBuffer can be assigned to more than one queue. These queues will compete for this PduRTxBuffer at runtime.

A Tx Buffer will be allocated by a queue to store a Pdu. The Tx Buffer is now not available for other queues until it is released again.

Tx Buffer sharing works for all shared buffer queues regardless if they are used by TP or IF routing paths.

2.7.3.1.2 Implicit Tx Buffer assignment

A PduRTxBuffer which is not referenced by any queue is used as implicit Tx Buffer. It can be used by any queue which has no explicit PduRTxBuffer assigned.

`PduRImplicitTxBufferRef` is automatically calculated by the PduR and shows all of the implicit Tx Buffers.



Shared Tx Buffer

All queues without explicit Tx Buffers will compete for the implicit PduRTxBuffers at runtime.

A Tx Buffer will be allocated by a queue to store a Pdu. The Tx Buffer is now not available for other queues until it is released again.

Tx Buffer sharing works for all shared buffer queues regardless if they are used by TP or IF routing paths.

2.7.3.2 Tx Buffer Length Configuration

2.7.3.2.1 Communication Interface

A Tx Buffer for communication interface queues must store the global Pdu length + optional meta data.

2.7.3.2.2 Transport Protocol

A Tx Buffer used for transport protocol queues must at least store the size of <TpThreshold> bytes + the optional meta data. This is the minimum requirement that this buffer can be used by this queue. It is not necessary to provide a buffer with the size equal to the Pdu length unless the TpThreshold is set to the Pdu length.



Note

Increase the PduRTxBuffer length (PduRPduMaxLength) in case the data rate on the destination is lower than the data rate of the source.

A buffer smaller than the routed I-PDU can result in wait-frames or a buffer-overflow if the destination connection is slower than the source connection. A buffer overflow can be avoided if the receiving TP connection allows dynamic adaptation of the block size (BS) (e.g. CanTp connection with BS greater 0). If a BS of 0 is used by some of the source TP connections, the configured buffer length should be dimensioned in a way that buffer overflows are avoided.



Caution

It is recommended to configure some small Tx buffers (e.g. 7 bytes) so that a single frame routing (e.g. functional request) does not occupy and block larger buffers. These shall be assigned to the corresponding queues.

2.7.3.3 Tx Buffer Selection Algorithm

The PduR uses the following rules to choose one of the configured Tx buffers:

Communication interface:

- ▶ The first buffer is used which is at least as large as the actual received Pdu length + metadata.

Transport protocol:

- ▶ If the size of the incoming I-PDU is smaller than the configured TP Threshold the smallest available buffer is used that can hold the entire I-PDU.
- ▶ If the size of the incoming I-PDU is larger than the configured TP Threshold it uses the smallest Tx buffer which can hold the entire I-PDU. If such a buffer is not available it will use the largest available buffer which has a size larger than the TpThreshold.
- ▶ If all buffer are occupied, the buffer request is rejected and the TP I-PDU is not routed.

2.7.4 Queue Depth

The correct queue depth largely depends on the expected frequency of communication and availability of other modules. An indication that the queue depth was set too small are queue overflows in case of queued communication interface and rejected routings in case of queued transport protocol routings. This behavior should be visible in integration tests.

2.8 Cross Partition Routing

2.8.1 General configuration

The general concept for cross partition routing is, that the Pdu must be queued and must be processed deferred on the destination partition.

To configure a cross partition routing path, each global Pdu (referred by PduRSrcPdu and PduRDestPdu) needs to be assigned to a /MICROSAR/EcuC/EcucPartitionCollection/EcucPartition via one of those parameters:

- ▶ /MICROSAR/EcuC/EcucPduCollection/Pdu/EcucPduDefaultPartitionRef
- ▶ /MICROSAR/EcuC/EcucPduCollection/Pdu/EcucPduDedicatedPartition



Note

EcucPduDedicatedPartition must only be used if the same global Pdu is used by different modules on different partitions.

If a global Pdu is only used by one module, the EcucPduDefaultPartitionRef can be used.

Either all Pdus need to be assigned to an EcucPartition or none of them.

The PduR MainFunction (/MICROSAR/PduR/PduRGeneral/PduRMainFunctionPeriod) must be configured and mapped to a corresponding task.

All deferred Pdus are processed in the MainFunction (either by using the deferred event cache or without using it). In addition, the multicore queue is processed.

The memory section configuration is described in chapter 3.3.2.

2.8.2 Multicore Queue

A specialized multicore queue (McQ) is used to handle the efficient routing between different Partitions/OsApplications. Compared to a regular PduR queue it is implemented lock-free and with less overhead. It is advised to use it if your configuration has many of the suitable routing paths.

A McQ is instantiated by creating /MICROSAR/PduR/PduRRoutingTables/PduRQueue/PduRMulticoreQueue. One container must be created per partition-tuple. Both involved partitions/OsApplications must be referred by the corresponding parameter in the PduRMulticoreQueue container. For each McQ container two unidirectional queues are automatically generated from OsApplication A to OsApplication B and vice versa.

The queue size can be set individually for each direction. This way the memory consumption can be reduced as there may be different throughput between different OsApplication tuples.

Each queue is shared between every routing path from partition A to partition B (and vice versa) which uses the McQ.

**Note**

The PduRMulticoreQueue container are created by a PduR validator. Only the queue sizes need to be set manually. The PduRMulticoreQueue itself must not be referenced by a PduRDestPdu.

An appropriate queue size is dependent on multiple factors regarding the Pdus routed by the multicore queue. Here, their size, quantity, the frequency of the routing and the periodicity of the PduR MainFunction are the key factors. The queue size can be different depending on the routing direction between Partitions (OsApplication A to OsApplication B and vice versa). A too small queue size is indicated by queue overflows of the multicore queue which leads to a queue flush. In this case all Pdus, which were currently in the queue are lost. If an overflow occurs the DET error "PDUR_E_MCQ_QUEUE_OVERFLOW" is reported.

Every entry to the queue has a few bytes overhead depending on the actual configuration and the size of some internal datatypes.

This queue can be used for the following routing paths:

- > Communication Interface Gateway routing without queue
- > Communication Interface Tx Api Forwarding with Data Provision Direct (with/without TxConfirmation)
- > Communication Interface Rx Api Forwarding

These routing paths must not reference a PduRQueue.

**Caution**

A regular PduR Queue must be used for routing paths with Data Provision Trigger Transmit and for all Transport Protocol routing paths. In this case a suitable PduRQueue must be referred.

**Note**

The multicore queue can also be used if there is only one core used, but different partitions are configured in the PduR.

2.8.3 Deferred Event Cache

It is strongly advised to use the Deferred Event Cache feature, if there are many cross-partition routing paths with a regular PduR queue. This reduces the runtime in the PduR_MainFunction significantly as only the actual destinations which need deferred processing are processed. Otherwise, a loop over all destinations must be used.

The deferred event cache can be activated by setting /MICROSAR/PduR/PduRGeneral/PduRDeferredEventCacheSize.

An appropriate queue size is dependent on the Pdus routed using the deferred event cache. Here, the quantity, frequency of the deferred routings and the periodicity of the PduR MainFunction are the key factors.

**Note**

Routing paths which are queued with the ‘multicore queue’ will not use the deferred event cache. Only routing paths with a regular PduR queue and deferred destination processing will use the deferred event cache.

**Caution**

If an overflow of the deferred event cache occurs, a fallback to a loop over all destinations is performed. This increases the runtime and shall be avoided by using a suitable size of the cache.

2.8.4 Spinlocks

For following kinds of routing paths, a regular PduR queue must be used if they shall be routed between cores:

- > Any kind of Transport Protocol routing
- > Communication Interface routing with Trigger Transmit data provision
- > Communication Interface routing which uses a regular queue and not the multicore queue (see 2.8.2).

The states of these routing paths must be protected with spinlocks if this routing path is a cross-core routing path. Cross-partition routing paths with the partitions assigned to the same core do not need spinlocks.

Shared Buffer may need spinlocks as well, as they might be allocated by multiple cores at the same time.

The PduR will create the needed lock variables which must be locked by the user. This must be done via the user-implemented Lock APIs:

- > Std_ReturnType Appl_TryToGetSpinlock(uint32* lockVar)
- > Std_ReturnType Appl_ReleaseSpinlock(uint32* lockVar)
- > uint32 Appl_GetSpinlockInitVal(void)

**Caution**

The spinlocks shall be created with the corresponding validator and shall not be changed manually.

Additional information regarding requirements for the implementation of the PduR Appl SpinLock APIs can be found in the Multi-Core UserManual [8] and the Doxygen documentation in the PduR source code.

2.8.4.1 SpinLock Retry Counter

Each spinlock has a user configurable retry counter (/MICROSAR/PduR/PduRGeneral/PduRSpinlockRetryCounter). While acquiring a spinlock this retry counter is incremented. If the counter reaches the configured value, the PduR assumes that the spinlock was acquired, even if TryToGetSpinlock returned with negative result. This is needed in case of Partitions/OsApplications of different ASIL and the spinlock variables were overwritten by random code. In this case the retry counter prevents a deadlock. The default value for the retry counter shall not be changed unless there are routings between Partitions/OsApplications of different ASIL.

If routings between different ASIL-Partitions exist, the value might have to be lowered to avoid excessively long blocking of the ASIL-Partition. In general, the value of the counter depends on many factors, for example the configuration and the used microcontroller. The appropriate value can be assessed by measuring the time of the call of Appl_TryToGetSpinlock. From there the number of retries can be calculated based on the maximum tolerable blocking time.



Caution

If the retry counter expires, a DET runtime error is reported. This is caused by memory corruption. This can be an indicator for an erroneous state of the system.

2.8.5 Restrictions

The following restrictions apply for the PduR Multi-Partition feature:

- ▶ Transport protocol routing paths are only supported with store-and-forward setting (Tp Threshold >= global Pdu length). This means the source of the routing must provide and copy all the data at once. Copying in multiple chunks is not supported.
- ▶ Communication interface routing paths with last-is-best behavior (TriggerTransmit data provision) have to use a SingleBuffer queue and are only supported if the Pdu is updated regularly by the source with a corresponding PduR_<SrcMsn>Transmit call. Without this API call the PduR will not update the data of the single buffer by itself. If a SingleBuffer queue is used, no TxConfirmation can be used for the transmission to the destination.
- ▶ Communication interface Tx routing paths via the multicore queue are considered successful if they could be written to the queue successfully. If, afterwards, the transmission on the destination core returns with negative result, this will not be propagated to the source module. This way the source module could expect a TxConfirmation which will never (for this transmission) be called. The entire data path shall be configured accordingly.
- ▶ CancelTransmit, CancelReceive and the ChangeParameter APIs are not supported between different cores.
- ▶ When using routing path groups, it must be ensured that all destinations are assigned to the same partition. The EnableRouting and DisableRouting APIs should only be called in the exact same partition context as the referenced destinations.

2.9 PduR in mixed ASIL Systems

The PduR provides the functionality to route Pdus between partitions of different ASIL.

2.9.1 Routing paths with safety related availability requirements

Routing paths with safety related availability requirements must:

- ▶ Only be a single-partition routing path (source and destination must be in the same ASIL-D partition). This is also required for all destinations and sources of 1:N, N:1 and N:M routing paths.
- ▶ Must not use a resource, which is used by any other multi-partition routing path or single-partition routing path on any other partition. This is especially important for the usage of PduRSharedBufferQueue. The used queue must not refer other PduRTxBuffer, which are used by other queues in multi-partition routing paths or single-partition routing path on any other partition.
- ▶ Not be in routing path groups, which contain other multi-partition routing paths. This routing path group must only be enabled or disabled on the same partition as the routing path is assigned to.

These requirements must be checked according to the safety manual of PduR.

2.9.2 Routing paths without safety requirements

All cross-partition routing paths must not have any safety requirements.

All RAM resources used by these routing paths will be mapped to shared memory, which can be accessed by all partitions. The shared memory can potentially be overwritten by arbitrary code from one of the lower-than-ASIL-D partitions. All accesses to RAM are sanitized so that overwritten RAM cannot influence routing paths with safety related availability requirements.

Regardless of the sanitation, this can result in following errors for unsafe routing paths:

- ▶ Pdu getting lost.
- ▶ Pdu data getting corrupted.
- ▶ Pdu data being sent to the wrong destination (it will not be sent to destinations of routing paths with safety related availability requirements).
- ▶ Other errors may occur.

3 Integration

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

3.1 Scope of Delivery

The delivery of the PDUR contains the files which are described in the chapters 3.1.1 and 3.1.2:

3.1.1 Static Files

File Name	Source Code Delivery	Description
PduR.c	■	This is the source file of the PDUR
PduR.h	■	This is the header file of PDUR
PduR_Bm.c	■	This is the source file of the PDUR Buffer Manager
PduR_Bm.h	■	This is the header file of the PDUR Buffer Manager
PduR_Fm.c	■	This is the source file of the PDUR Fifo Manager
PduR_Fm.h	■	This is the header file of the PDUR Fifo Manager
PduR_Lock.c	■	This is the source file of the PDUR Lock Manager
PduR_Lock.h	■	This is the header file of the PDUR Lock Manager
PduR_McQ.c	■	This is the source file of the PDUR Multicore Queue
PduR_McQ.h	■	This is the header file of the PDUR Multicore Queue
PduR_Rmlf.c	■	This is the source file of the PDUR Routing Manager If
PduR_Rmlf.h	■	This is the header file of the PDUR Routing Manager If
PduR_Rmlf_DeferredProcessing.c	■	This is the source file of the PDUR Routing Manager If Deferred Processing
PduR_Rmlf_DeferredProcessing.h	■	This is the header file of the PDUR Routing Manager If Deferred Processing
PduR_Rmlf_QueueProcessing.c	■	This is the source file of the PDUR Routing Manager If Queue Processing
PduR_Rmlf_QueueProcessing.h	■	This is the header file of the PDUR Routing Manager If Queue Processing
PduR_RmTp.c	■	This is the source file of the PDUR Routing Manager Tp
PduR_RmTp.h	■	This is the header file of the PDUR Routing Manager Tp

File Name	Source Code Delivery	Description
PduR_RmTp_TxInst.c	■	This is the source file of the PDUR Routing Manager Tp TxInstance
PduR_RmTp_TxInst.h	■	This is the header file of the PDUR Routing Manager Tp TxInstance
PduR_RmTp_TxRouting.c	■	This is the source file of the PDUR Routing Manager Tp Tx Routing
PduR_RmTp_TxRouting.h	■	This is the header file of the PDUR Routing Manager Tp Tx Routing
PduR_RmTp_RxInst.c	■	This is the source file of the PDUR Routing Manager Tp RxInstance
PduR_RmTp_RxInst.h	■	This is the header file of the PDUR Routing Manager Tp RxInstance
PduR_RmTp_RxRouting.c	■	This is the source file of the PDUR Routing Manager Tp Rx Routing
PduR_RmTp_RxRouting.h	■	This is the header file of the PDUR Routing Manager Tp Rx Routing
PduR_IFQ.c	■	This is the source file of the PDUR If Queue
PduR_IFQ.h	■	This is the header file of the PDUR If Queue
PduR_EvC.c	■	This is the source file of the PDUR Event Cache
PduR_EvC.h	■	This is the header file of the PDUR Event Cache
PduR_RPG.c	■	This is the source file of the PDUR Routingpath Group Manager
PduR_RPG.h	■	This is the header file of the PDUR Routingpath Group Manager
PduR_SingleBuffer.c	■	This is the source file of the PDUR Single Buffer
PduR_SingleBuffer.h	■	This is the header file of the PDUR Single Buffer
PduR_MultipleSourceHandler.c	■	This is the source file of the PDUR Multiple Source Handler
PduR_MultipleSourceHandler.h	■	This is the header file of the PDUR Multiple Source Handler
PduR_Report.c	■	This is the source file of the PDUR Reporting
PduR_Report.h	■	This is the header file of the PDUR Reporting
PduR_QAL.c	■	This is the source file of the PDUR Queue Abstraction Layer
PduR_QAL.h	■	This is the header file of the PDUR Queue Abstraction Layer
PduR_Mm.c	■	This is the source file of the PDUR Module Manager
PduR_Mm.h	■	This is the header file of the PDUR Module Manager

Table 3-1 Static files

3.1.2 Dynamic Files

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

File Name	Description
PduR_Cfg.h	<p>This file contains:</p> <ul style="list-style-type: none"> ▶ global constant macros ▶ global function macros ▶ global data types and structures ▶ global data prototypes ▶ global function prototypes <p>of CONFIG-CLASS PRE-COMPILe data.</p>
PduR_Lcfg.h	<p>This file contains:</p> <ul style="list-style-type: none"> ▶ global constant macros ▶ global function macros ▶ global data types and structures ▶ global data prototypes ▶ global function prototypes <p>of CONFIG-CLASS LINK data.</p>
PduR_Lcfg.c	<p>This file contains:</p> <ul style="list-style-type: none"> ▶ local constant macros ▶ local function macros ▶ local data types and structures ▶ local data prototypes ▶ local data ▶ global data <p>of CONFIG-CLASS LINK and PRE-COMPILe data.</p>
PduR_PBcfg.h	<p>This file contains:</p> <ul style="list-style-type: none"> ▶ global constant macros ▶ global function macros ▶ global data types and structures ▶ global data prototypes ▶ global function prototypes <p>of CONFIG-CLASS POST-BUILD data.</p>
PduR_PBcfg.c	<p>This file contains:</p> <ul style="list-style-type: none"> ▶ local constant macros ▶ local function macros ▶ local data types and structures ▶ local data prototypes ▶ local data ▶ global data <p>of CONFIG-CLASS POST-BUILD data.</p>
PduR_Types.h	This file contains types and defines for the PduR.
PduR_<Up>.h	This is the interface header of the PduR to an Upper Layer Module.
PduR_<Lo>.h	This is the interface header of the PduR to a Lower Layer Module

Table 3-2 Generated files

3.2 Critical Sections

The critical section PDUR_EXCLUSIVE_AREA_0 has to lock global interrupts to protect common critical sections.

3.3 Memory Sections

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

3.3.1 Memory sections for PduRTxBuffer

With PDUR_BUFFER_<SectionType> large Tx buffer can be mapped into an own memory section. By default, this is mapped into the Default Memory Section MSR_VAR_NO_INIT. If the mapping of the Memory Sections PDUR_BUFFER_<SectionType> should be changed, a /MICROSAR/MemMap/MemMapAllocation/MemMapSectionSpecificMapping must be configured in the MemMap Generator for the PDUR_BUFFER_<SectionType> Memory Sections and all PDUR_BUFFER_<SectionType> Memory sections of the PduR (/MICROSAR/PduR_Impl/ResourceConsumption/BUFFER_<SectionType>) must be referenced by /MICROSAR/MemMap/MemMapAllocation/MemMapSectionSpecificMapping/MemMapMemorySectionRef.

3.3.2 Memory sections for Multi-partition use case

If the PduR is used in a multi-partition use case it will put its data into Partition/OsApplication specific memory sections.

The user must ensure that partitions of different ASIL cannot write into each other's memory (RO: Read only access). For cross partition communication, the shared memory section must be accessible (RW: read and write access) by all involved partitions. See also Table 3-3.

Memory Section	OsApplicationX	OsApplicationY
PDUR_OsApplicationX_<SectionType>	RW	RO
PDUR_OsApplicationY_<SectionType>	RO	RW
PDUR_CommonSharedMemory_<SectionType>	RW	RW

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

For the mapping of the Partition/OsApplication specific memory sections Partition/OsApplication specific SwAddrMethods are used. For the OsApplication specific SwAddrMethods the mapping is done automatically by the Os. For the additional Partition specific memory section PDUR_SEC_CommonSharedMemory_<SectionType> the mapping of the SwAddrMethods must be done manually. In /MICROSAR/MemMap/MemMapAllocation/MemMapGenericMapping additional mappings must be configured, e.g. PDUR_CommonSharedMemory_CONST and PDUR_CommonSharedMemory_VAR. Here all Partition specific SwAddrMethods from the PduR must be referenced in the /MICROSAR/MemMap/MemMapAllocation/MemMapGenericMapping/MemMapSwAddressMethodRef. The partition specific SwAddrMethods of the PduR are all SwAddrMethods with

MICROSAR/PduR_ib_bswmd/SwAddrMethods/CommonSharedMemory_<SectionType>. Additionally, additional MICROSAR/MemMap/MemMapAddressingModeSet can be configured and these AddressingModeSets must be referenced from the configured mapping with the /MICROSAR/MemMap/MemMapAllocation/MemMapGenericMapping/MemMapAddressing ModeSetRef or an already existing fitting MemMapAddressingModeSet can be used.

**Caution**

In case of a mixed ASIL multi-partition configuration it must be ensured that the memory section which surrounds the PduR_ConfigDataPtr (PDUR_VAR_CLEARED_UNSPECIFIED) is solely writeable from the Partitions with the highest ASIL.

3.4 Type Definitions

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

Type Name	C-Type	Description	Value Range
PduR_RoutingPathGroupIdType	uint16	Identification of the routing path group. Routing path groups are defined in the PDU router configuration.	unsigned int

Table 3-4 Type definitions

4 API Description

4.1 Services provided by PDUR

4.1.1 PduR_PreInit

Prototype

```
void PduR_PreInit (const PduR_PBConfigType* ConfigPtr)
```

Parameter

ConfigPtr

- > NULL_PTR in the PDUR_CONFIGURATION_VARIANT_PRECOMPILE
- > Pointer to the PduR configuration data in the PDUR_CONFIGURATION_VARIANT_POSTBUILD_LOADABLE and PDUR_CONFIGURATION_VARIANT_POSTBUILD_SELECTABLE

Return code

void

none

Functional Description

This function pre initializes the PDU Router and performs configuration consistency checks. It initializes the shared memory which is used by all Partitions/OsApplications (in a multicore context).

If the initialization is performed successfully the PDU Router is in the state PduR_IsPreInitialized else not PduR_IsPreInitialized.

Particularities and Limitations

The function is used by the Ecu State Manager



Caution

PduR_PreInit shall not pre-empt any PDU Router function.

Call Context

See chapter 2.2 for details on the call context.

4.1.2 PduR_Init

Prototype

```
void PduR_Init (const PduR_PBConfigType* ConfigPtr)
```

Parameter

ConfigPtr	<ul style="list-style-type: none"> > NULL_PTR in the PDUR_CONFIGURATION_VARIANT_PRECOMPILE > Pointer to the PduR configuration data in the PDUR_CONFIGURATION_VARIANT_POSTBUILD_LOADABLE and PDUR_CONFIGURATION_VARIANT_POSTBUILD_SELECTABLE
-----------	--

Return code

void	none
------	------

Functional Description

This function initializes the PDU Router in the context of the calling Partition/OsApplication. If the initialization is performed successfully the PDU Router on this Partition/OsApplication is in the state PduR_IsInitialized else not PduR_IsInitialized.

Particularities and Limitations

The function is used by the Ecu State Manager



Caution

PduR_Init shall not pre-empt any PDU Router function.

Call Context

The function must be called on task level. To avoid problems calling the PDU Router module uninitialized it is important that the PDU Router module is initialized before interfaced modules.

Table 4-1 PduR_Init

4.1.3 PduR_InitMemory

Prototype

```
void PduR_InitMemory (void)
```

Parameter

void	none
------	------

Return code

void	none
------	------

Functional Description

The function initializes variables, which cannot be initialized with the startup code.

Particularities and Limitations

The function is called by the application.

Call Context

The function must be called on task level.

Table 4-2 PduR_InitMemory

4.2 Services

4.2.1 PduR_GetVersionInfo

Prototype	
<code>void PduR_GetVersionInfo (Std_VersionInfoType* versioninfo)</code>	
Parameter	
versioninfo	Pointer to where to store the version information of the PDU Router.
Return code	
void	none
Functional Description	
Returns the version information of the PDU Router.	
Particularities and Limitations	
The function is called by the application.	
Call Context	
The function can be called on interrupt and task level.	

Table 4-3 PduR_GetVersionInfo

4.2.2 PduR_GetConfigurationId

Prototype	
<code>uint32 PduR_GetConfigurationId (void)</code>	
Parameter	
void	none
Return code	
uint32	uint32
Functional Description	
Provides the unique identifier of the PDU Router configuration.	
Particularities and Limitations	
The function is called by the application.	
Call Context	
The function can be called on interrupt and task level.	

Table 4-4 PduR_GetConfigurationId

4.2.3 PduR_EnableRouting

Prototype	
<code>void PduR_EnableRouting (PduR_RoutingPathGroupIdType id)</code>	
Parameter	
id	Identification of the routing path group. Routing path groups are defined in the PDU router configuration.
Return code	
void	none
Functional Description	
This function enables a routing path group. If the routing path group does not exist or is already enabled, the function returns with no action.	
Particularities and Limitations	
The function is called by the BSW Mode Manager.	
Call Context	
The function can be called on interrupt and task level and has not to be interrupted by other PduR_EnableRouting or PduR_DisableRouting calls for the same id.	

Table 4-5 PduR_EnableRouting

4.2.4 PduR_DisableRouting

Prototype	
<code>void PduR_DisableRouting (PduR_RoutingPathGroupIdType id)</code>	
Parameter	
id	Identification of the routing path group. Routing path groups are defined in the PDU router configuration.
Return code	
void	none
Functional Description	
This function disables a routing path group. If the routing path group does not exist or is already disabled, the function returns with no action.	
Particularities and Limitations	
The function is called by the BSW Mode Manager.	
Call Context	
The function can be called on interrupt and task level and has not to be interrupted by other PduR_EnableRouting or PduR_DisableRouting calls for the same id.	

Table 4-6 PduR_DisableRouting

4.2.5 PduR_MainFunction

Prototype	
<pre>void PduR_MainFunction(void)</pre>	
Parameter	
-	-
Return code	
-	-
Functional Description	
<p>The PduR_MainFunction performs the deferred processing of (queued) routing paths.</p>	
Particularities and Limitations	
<ul style="list-style-type: none">> The PduR_PrelInit and PduR_Init have been called successfully.> The function is called cyclically by the BSW scheduler.	
Call context	
<ul style="list-style-type: none">> The function is called on task level.	

Table 4-7 PduR_MainFunction

4.3 Communication Interface

4.3.1 PduR_<GenericUp>Transmit

Prototype	
Std_ReturnType PduR_<GenericUp>Transmit (PduIdType id, PduInfoType* info)	
Parameter	
id	ID of the <GenericUp> I-PDU to be transmitted
info	Payload information of the I-PDU (pointer to data and data length)
Return code	
Std_ReturnType	Std_ReturnType E_OK The request was accepted by the PDU Router and by the destination layer. E_NOT_OK PduR_Init() has not been called or the id is not valid or the id is not forwarded in this identity or the info is not valid or the request was not accepted by the destination layer.
Functional Description	
The function serves to request the transmission of a Communication Interface I-PDU. The PDU Router evaluates the upper layer I-PDU handle and performs appropriate handle and port conversion. The call is routed to a lower communication interface module using the appropriate I-PDU handle of the destination layer.	
Particularities and Limitations	
The function is called by an upper layer.	
Call Context	
The function can be called on interrupt and task level and has not to be interrupted by other PduR_<GenericUp>Transmit calls for the same upper layer id.	

Table 4-8 PduR_<GenericUp>Transmit

4.3.2 PduR_<GenericLo>RxIndication

Prototype	
<pre>void PduR_<GenericLo>RxIndication (PduIdType id, const PduInfoType* info)</pre>	
Parameter	
id	Handle ID of the received <GenericLo> I-PDU.
info	Payload information of the received I-PDU (pointer to data and data length).
Return code	
void	none
Functional Description	
<p>The function is called by a lower communication interface to indicate the complete reception of a lower communication interface I-PDU. The PDU Router evaluates the lower communication interface I-PDU handle and performs appropriate handle and port conversion. The call is routed to an upper communication interface module using the appropriate I-PDU handle of the destination layer.</p>	
Particularities and Limitations	
<p>The function is called by a lower communication interface module.</p>	
Call Context	
<p>The function can be called on interrupt and task level and has not to be interrupted by other PduR_<GenericLo>RxIndication calls for the same a lower communication interface id.</p>	

Table 4-9 PduR_<GenericLo>RxIndication

4.3.3 PduR_<GenericLo>TriggerTransmit

Prototype	
<code>void PduR_<GenericLo>TriggerTransmit (PduIdType id, PduInfoType* info)</code>	
Parameter	
id	Handle ID of the <GenericLo> I-PDU that will be transmitted.
info	Contains a pointer to a buffer (SduDataPtr) to where the SDU data shall be copied, and the available buffer size in SduLength. On return, the service will indicate the length of the copied SDU data in SduLength.
Return code	
Std_ReturnType	E_OK: The SDU has been copied and the SduLength indicates the number of copied bytes. E_NOT_OK: No data has been copied, because PduR is not initialized or TxPdulId is not valid or PduInfoPtr is NULL_PTR or SduDataPtr is NULL_PTR or SduLength is too small.
Functional Description	
The function is called by a lower layer communication interface to request the TX I-PDU data before transmission. The PDU Router evaluates the lower layer communication interface I-PDU handle and performs appropriate handle and port conversion. The call is routed to an upper IF module using the appropriate I-PDU handles of the destination layer.	
Particularities and Limitations	
The function is called by a lower layer communication interface	
Call Context	
The function can be called on interrupt and task level and has not to be interrupted by other PduR_<GenericLo>TriggerTransmit calls for the same id.	

Table 4-10 PduR_<GenericLo>TriggerTransmit

4.3.4 PduR_<GenericLo>TxConfirmation

Prototype	
<code>void PduR_<GenericLo>TxConfirmation (PduIdType id)</code>	
Parameter	
id	Handle ID of the transmitted lower layer communication interface I-PDU.
Return code	
void	none
Functional Description	
<p>The function is called by a lower communication interface to confirm the complete transmission of a lower communication interface I-PDU. The PDU Router evaluates the lower communication interface I-PDU handle and performs appropriate handle and port conversion. The call is routed to an upper layer communication interface module using the appropriate I-PDU handle of the destination layer.</p>	
Particularities and Limitations	
<p>The function is called by a lower communication interface module.</p>	
Call Context	
<p>The function can be called on interrupt and task level and has not to be interrupted by other PduR_<GenericLo>TxConfirmation calls for the same lower layer communication interface id.</p>	

Table 4-11 PduR_<GenericLo>TxConfirmation

4.4 Transport Protocol

This chapter describes the interfaces provided to Upper Layers (e.g. Dcm, ApplTp and Cdds). Replace the tag <UpTp> by the MSN of the Upper Layer.

4.4.1 PduR_<GenericUpTp>ChangeParameter

Prototype	
<pre>Std_ReturnType PduR_<GenericUpTp>ChangeParameter (PduIdType id, TPParameterType parameter, uint16 value)</pre>	
Parameter	
id	ID of the <UpTp> I-PDU where the parameter has to be changed
parameter	The TP parameter that shall be changed.
value	The new value for the TP parameter.
Return code	
Std_ReturnType	<pre>Std_ReturnType E_OK: The parameter was changed successfully. E_NOT_OK: The parameter change was rejected.</pre>
Functional Description	
<p>The function serves to change a specific transport protocol parameter (e.g. block-size). The PDU Router evaluates the <UpTp> I-PDU handle and performs appropriate handle and port conversion. The call is routed to a lower TP module using the appropriate I-PDU handle of the destination layer.</p>	
Particularities and Limitations	
The function is called by <UpTp>.	
Call Context	
This function can be called on interrupt and task level and has not to be interrupted by other PduR_<UpTp>ChangeParameter calls for the same id.	

Table 4-12 PduR_<GenericUpTp>ChangeParameter

4.4.2 PduR_<GenericUpTp>CancelReceive

Prototype

```
Std_ReturnType PduR_<GenericUpTp>CancelReceive (PduIdType id)
```

Parameter

id	ID of the RX <GenericUp> I-PDU to be cancelled
----	--

Return code

Std_ReturnType	Std_ReturnType E_OK: Cancellation was executed successfully by the destination module. E_NOT_OK: Cancellation was rejected by the destination module.
----------------	---

Functional Description

The function serves to cancel the reception of a TP layer I-PDU.

The PDU Router evaluates the upper layer transport protocol I-PDU handle and performs appropriate handle and port conversion. The call is routed to a lower TP module using the appropriate I-PDU handle of the destination layer.

Particularities and Limitations

The function is called by an upper layer transport protocol module.

Call Context

The function can be called on interrupt and task level and has not to be interrupted by other PduR_<GenericUpTp>CancelReceive calls for the same upper layer id.

Table 4-13 PduR_<GenericUpTp>CancelReceive

4.4.3 PduR_<GenericUpTp>CancelTransmit

Prototype

```
Std_ReturnType PduR_<GenericUpTp>CancelTransmit (PduIdType id)
```

Parameter

id	ID of the TX upper layer I-PDU of the routing that must be cancelled in the lower layer
----	---

Return code

Std_ReturnType	Std_ReturnType E_OK The cancellation request was accepted by the PDU Router and by the TP layer. E_NOT_OK PduR_Init() has not been called or the id is not valid or the id is not forwarded in this identity or the request was not accepted by the TP layer.
----------------	--

Functional Description

The function serves to cancel the transmission of a TP layer I-PDU.

The PDU Router evaluates the upper layer transport protocol I-PDU handle and performs appropriate handle and port conversion. The call is routed to a lower TP module using the appropriate I-PDU handle of the destination layer.

Particularities and Limitations

The function is called by an upper layer transport protocol module

Call Context

The function can be called on interrupt and task level and has not to be interrupted by other PduR_<GenericUpTp>CancelTransmit calls for the same upper layer id.

Table 4-14 PduR_<GenericUpTp>CancelTransmit

4.4.4 PduR_<GenericLoTp>StartOfReception

Prototype

```
BufReq_ReturnType PduR_<GenericLoTp>StartOfReception (PduIdType id,
PduInfoType info, PduLengthType TpSduLength, PduLengthType*
bufferSizePtr)
```

Parameter

id	ID of the <GenericLo> I-PDU that will be received.
info	Pointer to the buffer (SduDataPtr) contains MetaData if this feature is enabled.
TpSduLength	Length of the entire <GenericLo> TP SDU which will be received
bufferSizePtr	Pointer to the receive buffer in the receiving module. This parameter will be used to compute Block Size (BS) in the transport protocol module.

Return code

BufReq_ReturnType	BufReq_ReturnType BUFREQ_OK Connection has been accepted. bufferSizePtr indicates the available receive buffer. BUFREQ_E_NOT_OK PduR_Init() has not been called or the id is not valid or the id is not forwarded in this identity or the info is not valid or the request was not accepted by the upper layer. or no buffer is available
-------------------	--

Functional Description

This function will be called by the lower layer at the start of receiving an I-PDU. The I-PDU might be fragmented into multiple N-PDUs (FF with one or more following CFs) or might consist of a single N-PDU (SF).

Particularities and Limitations

The function is called by lower layer transport protocol module.

Call Context

The function can be called on interrupt and task level and has not to be interrupted by other PduR_<GenericLoTp>StartOfReception calls for the same lower layer transport protocol id.

Table 4-15 PduR_<GenericLoTp>StartOfReception

4.4.5 PduR_<GenericLoTp>CopyRxData

Prototype

```
BufReq_ReturnType PduR_<GenericLoTp>CopyRxData (PduIdType id,
PduInfoType* info, PduLengthType* bufferSizePtr)
```

Parameter

id	ID of the lower layer transport protocol I-PDU that will be received.
info	Pointer to the buffer (SduDataPtr) and its length (SduLength) containing the data to be copied by PDU Router module in case of gateway or upper layer module in case of reception.
bufferSizePtr	Available receive buffer after data has been copied.

Return code

BufReq_ReturnType	BufReq_ReturnType BUFREQ_OK Buffer request accomplished successful. BUFREQ_E_NOT_OK PduR_Init() has not been called or the id is not valid or the id is not forwarded in this identity or the info is not valid or the request was not accepted by the upper layer. or the request length to copy is greater than the remaining buffer size BUFREQ_E_OVFL The upper TP module is not able to receive the number of bytes. The request was not accepted by the upper layer.
-------------------	--

Functional Description

This function is called by the lower layer transport protocol if data has to be copied to the receiving module. Parallel routing of several I-PDU is possible.

Particularities and Limitations

The function is called by lower layer transport protocol

Call Context

The function can be called on interrupt and task level and has not to be interrupted by other PduR_<GenericLoTp>CopyRxData calls for the same lower layer transport protocol id.

Table 4-16 PduR_<GenericLoTp>CopyRxData

4.4.6 PduR_<GenericLoTp>CopyTxData

Prototype

```
BufReq_ReturnType PduR_<GenericLoTp>CopyTxData (PduIdType id,
PduInfoType* info, RetryInfoType* retry, PduLengthType*
availableDataPtr)
```

Parameter

id	ID of the lower layer I-PDU that will be transmitted.
info	Pointer to the destination buffer and the number of bytes to copy. In case of gateway the PDU Router module will copy otherwise the source upper layer module will copy the data. If not enough transmit data is available, no data is copied. The transport protocol module will retry. A size of copy size of "0" can be used to indicate state changes in the retry parameter.
retry	retry not supported yet, is always a NULL_PTR.
availableDataPtr	Indicates the remaining number of bytes that are available in the PDU Router Tx buffer.

Return code

BufReq_ReturnType	BufReq_ReturnType BUFREQ_OK The data has been copied to the transmit buffer successful. BUFREQ_E_NOT_OK PduR_Init() has not been called or the id is not valid or the id is not forwarded in this identity or the info is not valid or the request was not accepted by the upper layer and no data has been copied. BUFREQ_E_BUSY The request cannot be processed because the TX data is not available and no data has been copied. The TP layer might retry later the copy process.
-------------------	---

Functional Description

This function is called by a lower layer transport protocol module to query the transmit data of an I-PDU segment.

Particularities and Limitations

The function is called by a lower layer transport protocol module.

Call Context

The function can be called on interrupt and task level and has not to be interrupted by other PduR_<GenericLoTp>CopyTxData calls for the same a lower layer transport protocol module id.

Table 4-17 PduR_<GenericLoTp>CopyTxData

4.4.7 PduR_<GenericLo>TpTxConfirmation

Prototype

```
void PduR_<GenericLo>TpTxConfirmation (PduIdType id, Std_ReturnType result)
```

Parameter

id	ID of the <GenericLo> I-PDU that will be transmitted.
result	<p>Result of the TP transmission</p> <p>E_OK The TP transmission has been completed successfully.</p> <p>E_NOT_OK PduR_Init() has not been called or the transmission was aborted or the id is not valid or the id is not forwarded or the request was not accepted by the destination upper layer.</p>

Return code

void	none
------	------

Functional Description

The function is called by a lower layer transport protocol module to confirm a successful transmission of a lower layer transport protocol module TX SDU or to report an error that occurred during transmission. The PDU Router evaluates the lower layer transport protocol module I-PDU handle and performs appropriate handle and port conversion. The call is routed to an upper TP module using the appropriate I-PDU handle of the destination layer.

Particularities and Limitations

The function is called by a lower layer transport protocol module.

Call Context

The function can be called on interrupt and task level and has not to be interrupted by other PduR_<GenericLo>TpTxConfirmation calls for the same the lower layer transport protocol module id.

Table 4-18 PduR_<GenericLo>TpTxConfirmation

4.4.8 PduR_<GenericLo>TpRxIndication

Prototype

```
void PduR_<GenericLo>TpRxIndication (PduIdType id, Std_ReturnType result)
```

Parameter

id	ID of the <GenericLo> I-PDU that will be received.
result	<p>Result of the TP reception</p> <p>E_OK The TP reception has been completed successfully.</p> <p>E_NOT_OK PduR_Init() has not been called</p> <p> or the reception was aborted</p> <p> or the id is not valid</p> <p> or the id is not forwarded</p> <p> or the request was not accepted by the destination upper layer.</p>

Return code

void	none
------	------

Functional Description

The function is called by the lower layer transport protocol module to indicate the complete reception of a lower layer transport protocol module SDU or to report an error that occurred during reception. The PDU Router evaluates the lower layer transport protocol module I-PDU handle and performs appropriate handle and port conversion. The call is routed to an upper TP module using the appropriate I-PDU handle of the destination layer.

Particularities and Limitations

The function is called by a lower layer transport protocol module.

Call Context

The function can be called on interrupt and task level and has not to be interrupted by other PduR_<GenericLo>TpRxIndication calls for the same lower layer transport protocol module.

Table 4-19 PduR_<GenericLo>TpRxIndication

4.4.9 PduR_<GenericUpTp>Transmit

Prototype	
Std_ReturnType PduR_<GenericUpTp>Transmit (PduIdType id, PduInfoType* info)	
Parameter	
id	ID of the upper layer I-PDU that have to be transmitted
info	Payload information of the I-PDU (pointer to data and data length)
Return code	
Std_ReturnType	Std_ReturnType E_OK The request was accepted by the PDU Router and by the destination layer. E_NOT_OK PduR_Init() has not been called or the id is not valid or the id is not forwarded in this identity or the info is not valid or the request was not accepted by the TP layer.
Functional Description	
<p>The function serves to request the transmission of a TP layer I-PDU.</p> <p>The PDU Router evaluates the incoming I-PDU ID and performs appropriate ID and port conversion. The call is routed to the TP layer using the appropriate I-PDU handle of the destination layer.</p>	
Particularities and Limitations	
The function is called by an upper layer transport protocol module.	
Call Context	
The function can be called on interrupt and task level and has not to be interrupted by other PduR_<GenericUpTp>Transmit calls for the same an upper layer transport protocol module id.	

Table 4-20 PduR_<GenericUpTp>Transmit

4.5 Service Ports

4.5.1 Complex Device Driver Interaction

Besides the AUTOSAR modules, Complex Device Drivers (CDD) are also possible as upper layer and lower transport protocol or communication interface modules for the PduR. When a callout function of the PduR is invoked from a lower or upper layer module for a PDU that is transmitted or received by a CDD, the PduR invokes the corresponding target function of the CDD. If all PDUs transmitted or received by a CDD are referenced by communication interface modules, the CDD requires a communication interface API. If all PDUs transmitted or received by a CDD are referenced by transport protocol modules, the CDD requires a transport protocol API.

A CDD can either require a communication interface API or it can require a transport protocol API but not both. The API functions provided by the PduR for the CDD interaction contain the CDD's name.

5 Configuration

5.1 Use Case Configuration: Communication interface range gateway

The PduR routing paths configuration is typically focused on the routing of dedicated Pdus and their bus-specific representation (e.g. the concrete CAN ID, DLC, ...). For some network and communication architectures it might be helpful to avoid the dedicated configuration of all possible and required PduR routing paths. Especially if a huge amount of Pdus shall be routed between networks, the PduR routing paths configuration gets extensive, error-prone and requires a lot of hardware resources (ROM / RAM).

To overcome this situation, so called range-routings could be used. For the routing of a full Pdu range between two networks just single PduR routing path needs to be configured.

Technically, the range-routing is based on the usage of MetaData information appended to the Pdu data field. During Pdu reception the related bus interface module stores all required Pdu meta information (the CAN ID) in the MetaData part of the Pdu data field. The PduR itself performs the routing of the full Pdu data field, including the MetaData. During transmission the related bus interface module uses the MetaData information for dynamic adaptation (the CAN ID) of the transmitted bus Pdu. The range of the routed Pdus is defined by the bus interface specific Rx Pdu range. In case of CanIf, the CAN ID Rx filter is used to define the range. Figure 5-1 visualizes the range-routing technique.

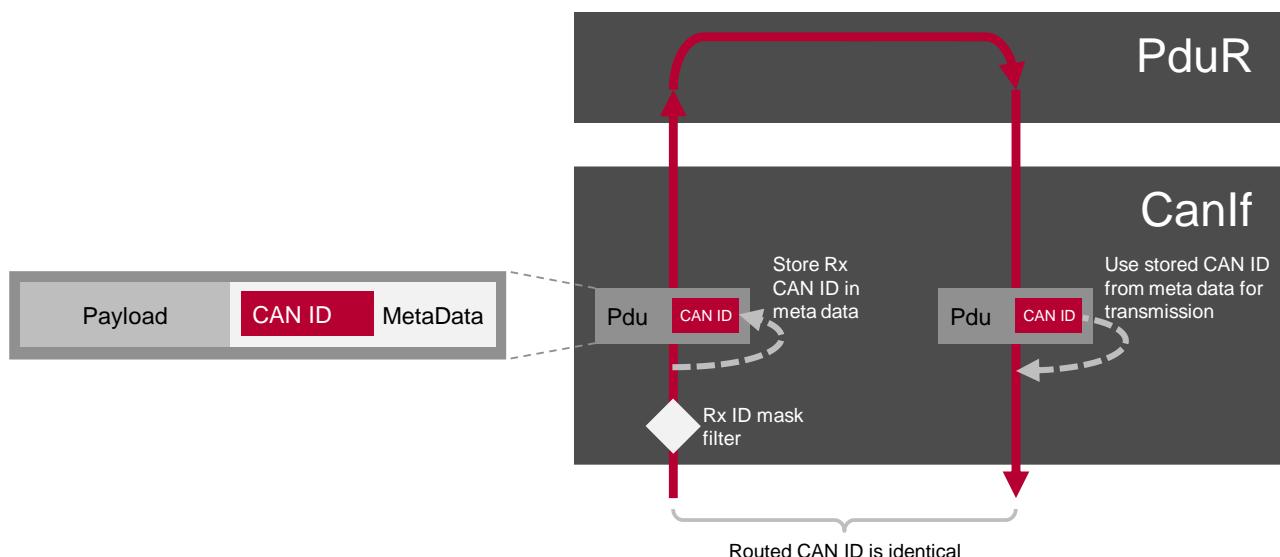


Figure 5-1 Meta data routing with CanIf



Limitations

There are some limitations when using the described range-routing technique:

- ▶ Available for CAN only
- ▶ No CAN ID conversion possible
- ▶ No length conversion possible
- ▶ No dynamic PDU lengths possible

5.1.1 Step-by-step configuration



Configuration Example

All following step-by-step instructions are based on this range-routing example:

CAN ID code	CAN ID mask	DLC	Source network / channel	Destination network(s) / channel(s)
0x700	0x708	8	CAN0	CAN1, CAN2
0x708	0x708	8	CAN1	CAN0
0x708	0x708	8	CAN2	CAN0

Table 5-1 Example range routings

The CAN ID range is defined by the condition

<received CAN-ID> & <mask> == <code> & <mask>

The following step-by-step configurations steps will result in the following CanIf / PduR range routing architecture shown in Figure 5-2.

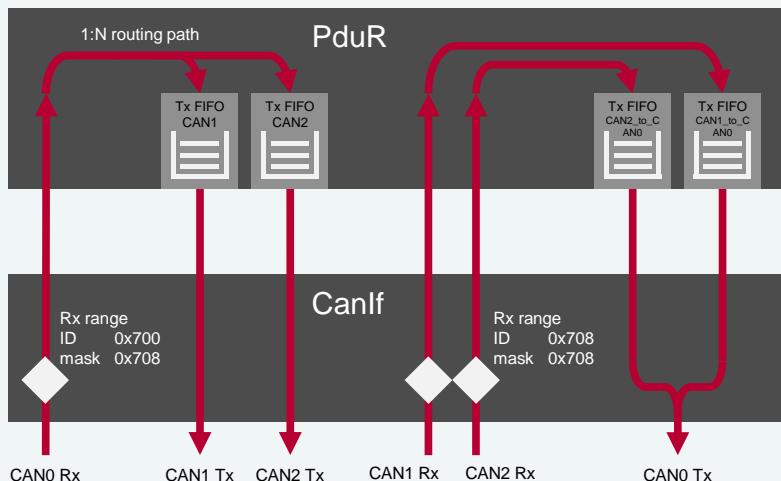


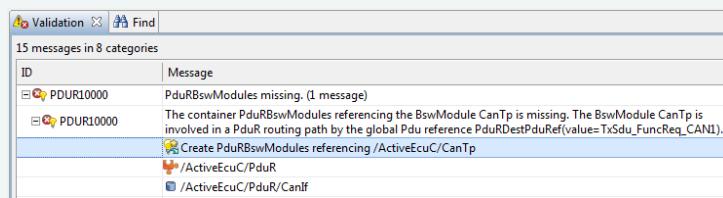
Figure 5-2 CanIf / PduR range routing example overview



Derived model elements / Validation and solving actions

During project setup the EcuC configuration will be automatically derived from the provided input files. Therefore some of the following manual configuration steps are redundant. In this case, please extend or adapt the existing configuration containers and parameters to the described step-by-step configuration.

Parameters and containers which will be created and configured by background-validations are not described explicitly. Please finalize all manual configuration steps before solving the open validations results. Use the provided solving actions.



Create global Pdus

- ▶ Create global Pdus for every required range routing source and destination channel.
Global Pdu container: /MICROSAR/EcuC/EcucPduCollection/Pdu
- ▶ Create and configure a MetaData length. 2 bytes are required for standard CAN IDs, 4 bytes are required for extended CAN identifiers.
MetaData length: /MICROSAR/EcuC/EcucPduCollection/Pdu/MetaDataLength
- ▶ Configure the Pdu length to the Pdu payload length.
Pdu length: /MICROSAR/EcuC/EcucPduCollection/Pdu/PduLength

For the example configuration this results in the following global Pdu configuration:

The screenshot shows a software interface for managing ECU configurations. On the left, there's a tree view under 'EcuC' with nodes like 'EcucGeneral' and 'EcucPduCollection'. Under 'EcucPduCollection', there's a folder named 'Pdus' containing several Pdu entries. On the right, a table lists these Pdus with their properties. A red box highlights the first seven rows of the table, which correspond to the RxRangePdu entries mentioned in the text.

Pdus	Meta Data Length	Pdu Length [Byte]
RxRangePdu_CAN0	2	8
RxRangePdu_CAN1	2	8
RxRangePdu_CAN2	2	8
TxRangePdu_CAN0_to_CAN1	2	8
TxRangePdu_CAN0_to_CAN2	2	8
TxRangePdu_CAN1_to_CAN0	2	8
TxRangePdu_CAN2_to_CAN0	2	8



Create CanIf Rx Pdus

- ▶ Create a CanIf Rx Pdu for every required range routing source channel.
CanIf Rx Pdu container: /MICROSAR/CanIf/CanIfInitCfg/CanIfRxPduCfg
- ▶ Configure the Rx CAN ID range by the CAN ID code and the CAN ID mask
/MICROSAR/CanIf/CanIfInitCfg/CanIfRxPduCfg/CanIfRxPduCanId
/MICROSAR/CanIf/CanIfInitCfg/CanIfRxPduCfg/CanIfRxPduCanIdMask
The CAN ID range is defined by the filter condition
<received CAN-ID> & <mask> == <code> & <mask>
- ▶ Configure the CAN ID type (standard or extended) and the DLC with the parameters
/MICROSAR/CanIf/CanIfInitCfg/CanIfRxPduCfg/CanIfRxPduCanIdType
/MICROSAR/CanIf/CanIfInitCfg/CanIfRxPduCfg/CanIfRxPduDlc
- ▶ Reference the related channel specific Rx global Pdu created in the previous steps with the parameter
/MICROSAR/CanIf/CanIfInitCfg/CanIfRxPduCfg/CanIfRxPduRef
- ▶ Reference the related CAN channel hardware receive object (HRH) used for reception of the range Pdus with the parameter
/MICROSAR/CanIf/CanIfInitCfg/CanIfRxPduCfg/CanIfRxPduHrhIdRef
- ▶ Assign the PduR as upper layer user with the parameter /MICROSAR/CanIf/CanIfInitCfg/CanIfRxPduCfg/CanIfRxPduUserRxIndicationUL

For the example configuration this results in the following CanIf Rx Pdu configuration:

CanIfRxPduCfgs > CanIfRxRangePdu_CAN0, CanIfRxRangePdu_CAN1, CanIfRxRangePdu_CAN2								
	Rx Pdu Can Id	Rx Pdu Can Id Mask	Rx Pdu Can Id Type	Rx Pdu Dlc	Rx Pdu Dlc Check	Rx Pdu Ref	Rx Pdu Hrh Id Ref	Rx Pdu User Rx Indication UL
+ CanIfRxRangePdu_CAN0	0x700	0x708	STANDARD_CAN	8	<input checked="" type="checkbox"/>	* RxRangePdu_CAN0	CN_Rx_CAN0	PDUR
CanIfRxRangePdu_CAN1	0x708	0x708	STANDARD_CAN	8	<input checked="" type="checkbox"/>	* RxRangePdu_CAN1	CN_Rx_CAN1	PDUR
CanIfRxRangePdu_CAN2	0x708	0x708	STANDARD_CAN	8	<input checked="" type="checkbox"/>	* RxRangePdu_CAN2	CN_Rx_CAN2	PDUR



Create CanIf Tx Pdus

- ▶ Create a CanIf Tx Pdu for every required range routing destination channel.
CanIf Tx Pdu container: /MICROSAR/CanIf/CanIfInitCfg/CanIfTxPduCfg
- ▶ Configure the Tx CAN ID used for prioritization of this dynamic Tx Pdu against other static Pdus with the parameter
/MICROSAR/CanIf/CanIfInitCfg/CanIfTxPduCfg/CanIfTxPduCanId
Use the highest priority CAN ID of the routing range for this prioritization ID.
- ▶ Configure the CAN DLC with the parameter
/MICROSAR/CanIf/CanIfInitCfg/CanIfTxPduCfg/CanIfTxPduDlc
- ▶ Reference the related global Pdu created in the previous steps with the parameter
/MICROSAR/CanIf/CanIfInitCfg/CanIfTxPduCfg/CanIfTxPduRef
- ▶ Reference the related channel specific CanIf Tx buffer object used for transmission of the range Pdus with the parameter
/MICROSAR/CanIf/CanIfInitCfg/CanIfTxPduCfg/CanIfTxPduBufferRef

For the example configuration this results in the following CanIf Tx Pdu configuration:

CanIfTxPduCfgs > CanIfTxRangePdu_CAN0_to_CAN1, CanIfTxRangePdu_CAN0_to_CAN2, CanIfTxRangePdu_CAN1_to_CAN0, CanIfTxRangePdu_CAN1_to_CAN1					
	Tx Pdu Can Id	Tx Pdu Dlc [Byte]	Tx Pdu Ref	Tx Pdu Buffer Ref	
CanIfTxRangePdu_CAN0_to_CAN1	0x700	8	TxRangePdu_CAN0_to_CAN1	CanIfBufferCfg_Tx_CAN1	
CanIfTxRangePdu_CAN0_to_CAN2	0x700	8	TxRangePdu_CAN0_to_CAN2	CanIfBufferCfg_Tx_CAN2	
CanIfTxRangePdu_CAN1_to_CAN0	0x700	8	TxRangePdu_CAN1_to_CAN0	CanIfBufferCfg_Tx_CAN0	
CanIfTxRangePdu_CAN2_to_CAN0	0x700	8	TxRangePdu_CAN2_to_CAN0	CanIfBufferCfg_Tx_CAN0	



Create PduR routing paths

Finally create the PduR routing paths connecting the CanIf Rx and Tx range Pdus. A single PduR routing path for every channel specific range routing is required.

- ▶ **Create a PduR routing path:**
/MICROSAR/PduR/PduRRoutingTables/PduRRoutingTable/PduRRoutingPath
- ▶ **Reference the related channel specific Rx global Pdu at the routing path source:**
/MICROSAR/PduR/PduRRoutingTables/PduRRoutingTable/PduRRoutingPath/PduRSrcPdu/PduRSrcPduRef
- ▶ **Reference the related channel specific Tx global Pdu at the routing path destination:**
/MICROSAR/PduR/PduRRoutingTables/PduRRoutingTable/PduRRoutingPath/PduRDestPdu/PduRDestPduRef

For the example configuration this results in the following PduR routing path configuration:

RoutingPath	RoutingPath Source Global Pdu	RoutingPath Destination(s) Global Pdu(s)
RP_RangeRouting_CAN0_to_CAN1_2	RxRangePdu_CAN0	TxRangePdu_CAN0_to_CAN1, TxRangePdu_CAN0_to_CAN2
RP_RangeRouting_CAN1_to_CAN0	RxRangePdu_CAN1	TxRangePdu_CAN1_to_CAN0
RP_RangeRouting_CAN2_to_CAN0	RxRangePdu_CAN2	TxRangePdu_CAN2_to_CAN0

The screenshot shows the 'PduR' section of the tool. On the left is a tree view of PduR components. On the right, a form is displayed for a specific routing path. The 'Src Pdu Ref' field contains 'RxRangePdu_CAN0' and is highlighted with a red box.

The screenshot shows the 'PduRDestPdus' section for the 'RP_RangeRouting_CAN0_to_CAN1_2' path. It lists two entries: 'Dest_TxRangePdu_CAN1' and 'Dest_TxRangePdu_CAN2'. Both entries in the 'Dest Pdu Ref' column are highlighted with a red box.

For further details regarding the PduR communication interface gateway, please refer to chapter 2.5.

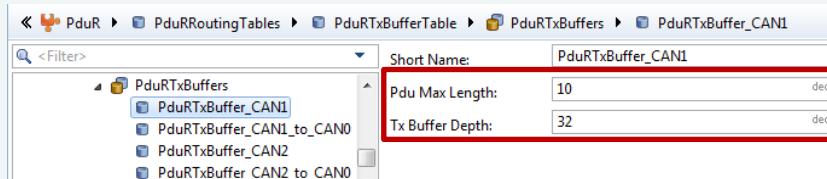
5.1.2 Optional configuration variants / options



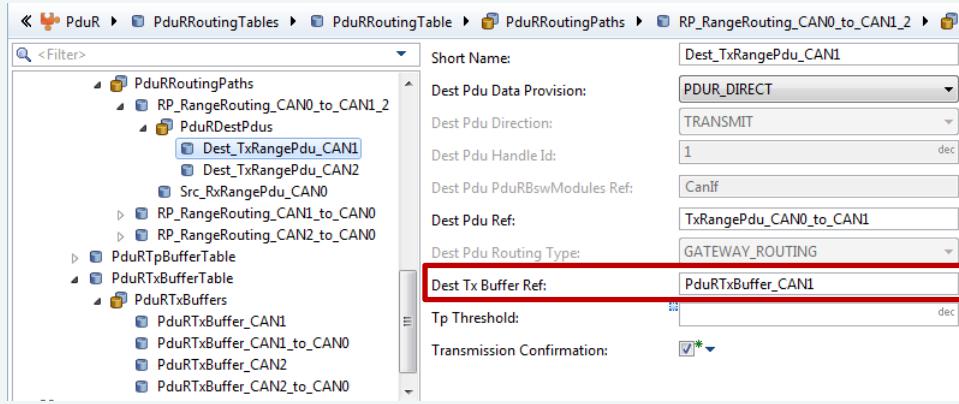
[Optional] Configure PduR FIFO routing

In case the sequence of successive routed range Pdus shall be retained, a FIFO queue must be configured within the PduR.

- ▶ Create PduR Tx buffer (FIFO) for every range routing destination channel:
`/MICROSAR/PduR/PduRRoutingTables/PduRTxBufferTable/PduRTxBuffer`
- ▶ Configure the Tx buffer length to the length of the routed global Pdu (including the MetaData length):
`/MICROSAR/PduR/PduRRoutingTables/PduRTxBufferTable/PduRTxBuffer/PduRPduMaxLength`
- ▶ Configure the Tx buffer depth to the maximum expected amount of queued Pdus (see also chapter 2.5.2):
`/MICROSAR/PduR/PduRRoutingTables/PduRTxBufferTable/PduRTxBuffer/PduRTxBufferDepth`



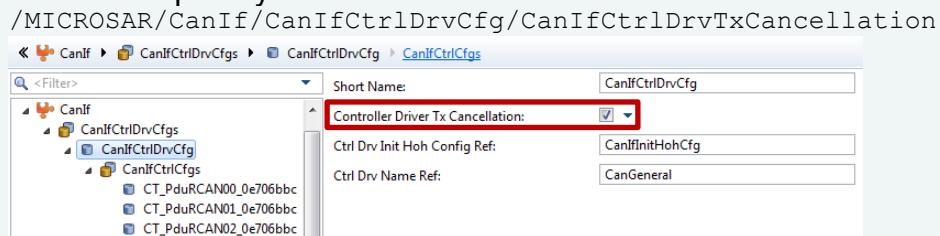
- ▶ Reference the created Tx buffers at the related PduR range routing paths:
`/MICROSAR/PduR/PduRRoutingTables/PduRRoutingTable/PduRRoutingPath/PduRDestPdu/PduRDestTxBufferRef`





[Optional] CAN priority inversion

- ▶ Enable the CanIf feature „Cancellation of PDUs and requeueing“ in order to avoid inner priority inversion:



- ▶ Enable the CAN driver feature „Multiplexed Transmission“ in order to avoid external priority inversion:



If this feature is enabled, multiple hardware objects are used for transmission of a single logical Tx object.

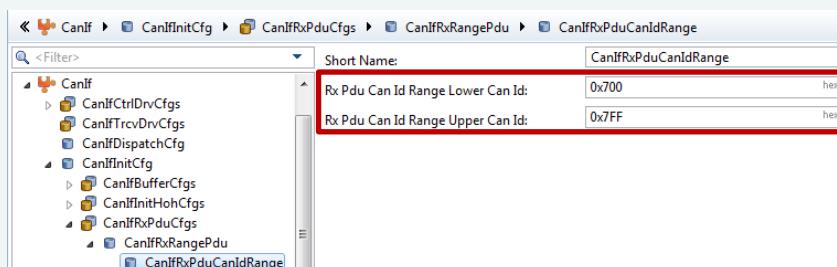
Hint: These features are not supported by all CAN controllers. Please refer to [5] and [6].



[Optional] Alternative CanIf Rx range configuration method

Depending on the required CAN ID range, the range can also be configured using an upper- and lower layer ID using the following container / parameters:

```
/MICROSAR/CanIf/CanIfInitCfg/CanIfRxPduCfg/CanIfRxPduCanIdRange
/MICROSAR/CanIf/CanIfInitCfg/CanIfRxPduCfg/CanIfRxPduCanIdRange/CanIfRxPduCanIdRangeLowerCanId
/MICROSAR/CanIf/CanIfInitCfg/CanIfRxPduCfg/CanIfRxPduCanIdRange/CanIfRxPduCanIdRangeUpperCanId
```



In case of this alternative range configuration method, the previously used range configuration parameters must not be used:

```
/MICROSAR/CanIf/CanIfInitCfg/CanIfRxPduCfg/CanIfRxPduCanId
/MICROSAR/CanIf/CanIfInitCfg/CanIfRxPduCfg/CanIfRxPduCanIdMask
```

For further details about the CanIf and CAN driver modules, please refer to [5] and [6].

5.2 Use Case Configuration: Functional requests gateway routing

Gateway ECUs typically include diagnostic services, handling physical and functional addressed requests used by external diagnostic tools during the development, manufacturing and service.

Functionally addressed request messages are a kind of broadcast requests which shall be processed by all (or a dedicated set of) ECUs. Typically this includes the gateway ECU itself as well. In case the ECUs are distributed on multiple CAN networks, the gateway ECU needs to handle the broadcasting of the request messages to the connected sub-network as well as the handling of the functional requests by the gateway-own diagnostic handler.

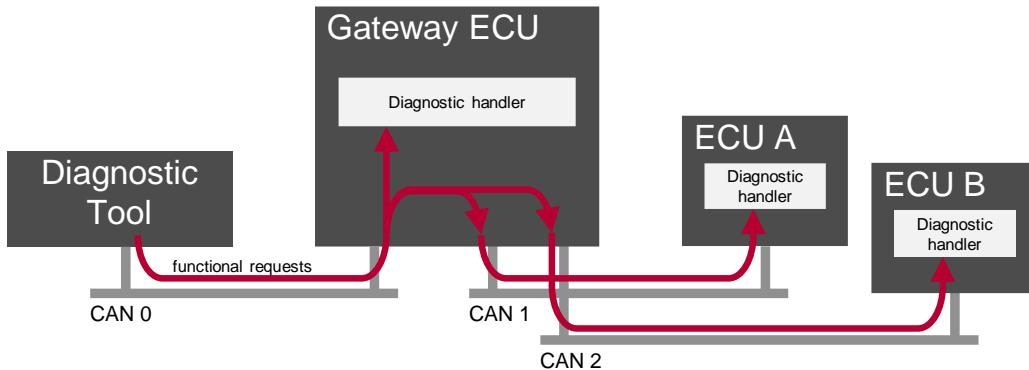


Figure 5-3 Example functional requests gateway network architecture

To realize the gateway behavior as visualized in Figure 5-3, a PduR 1:N transport protocol gateway could be configured, as shown in Figure 5-4.

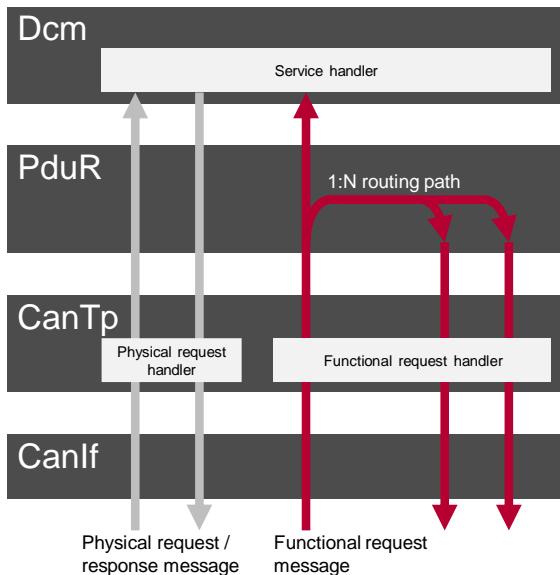


Figure 5-4 Functional request gateway architecture



Handling of physically addressed diagnostic messages

Routing of physically addressed diagnostic messages is not focus of this chapter.
Please refer to chapter 5.1 introducing range-routing paths which could be used for efficient and simple routing of physically addressed diagnostic messages.

5.2.1 Step-by-step configuration



Configuration Example

All following step-by-step instructions are based on the following functional diagnostic routing example:

Functional diagnostic request ID	DLC	Source network / channel	Destination network(s) / channel(s)
0x7DF	8	CAN0	CAN1, CAN2

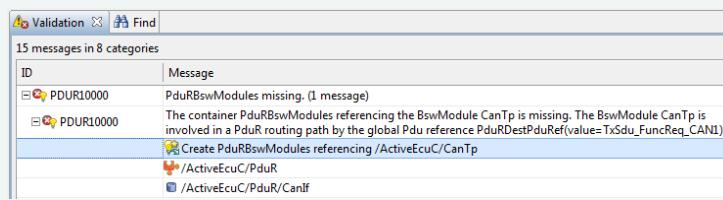
Table 5-2 Example functional diagnostic request routing



Derived model elements / Validation and solving actions

During project setup the EcuC configuration will be automatically derived from the provided input files. Therefore some of the following manual configuration steps are redundant. In this case, please extend or adapt the existing configuration containers and parameters to the described step-by-step configuration.

Parameters and containers which will be created and configured by background-validations are not described explicitly. Please finalize all manual configuration steps before solving the open validations results. Use the provided solving actions.



Create global Pdus / Sdus

- ▶ Create two global Pdus for the received functional request. The first global Pdu interconnects the CanIf and CanTp receive paths. The second global Pdu (Sdu) interconnects the CanTp, PduR and the related diagnostic handler (e.g. Dcm).
Global Pdu container: /MICROSAR/EcuC/EcucPduCollection/Pdu
This step is optional if the own functional request routing path was automatically derived based on input files during project setup.
- ▶ Create a pair of global Pdus (Pdu and Sdu) for every destination channel where the functional requests shall be routed to.
- ▶ Configure the Pdu length to the CAN frame length of the functional request message. The length of the Sdu (interconnection between CanTp, PduR and the related diagnostic handler) shall be the length of transport protocol payload.
Pdu length: /MICROSAR/EcuC/EcucPduCollection/Pdu/PduLength

For the example configuration this results in the following global Pdu configuration:

The screenshot shows the EcuC configuration interface with the following details:

- Left sidebar: EcuC > EcucPduCollection > Pdus > RxPdu_FuncReq_CAN0, TxPdu_FuncReq_CAN1, TxPdu_FuncReq_CAN2, RxSdu_Fu
- Table view titled "Pdus" with columns: Pdu, Pdu Length [Byte], Meta Data Length.
- Data rows (highlighted with a red box):

Pdu	Pdu Length [Byte]	Meta Data Length
RxPdu_FuncReq_CAN0	8	
RxSdu_FuncReq_CAN0	7	
TxPdu_FuncReq_CAN1	8	
TxPdu_FuncReq_CAN2	8	
TxSdu_FuncReq_CAN1	7	
TxSdu_FuncReq_CAN2	7	



Create CanIf Rx Pdus

The following steps are optional if the own functional request routing path was automatically derived based on input files during project setup.

- ▶ Create a CanIf Rx Pdu for the functional diagnostic request message.
CanIf Rx Pdu container: /MICROSAR/CanIf/CanIfInitCfg/CanIfRxPduCfg
- ▶ Configure the static Rx CAN ID of the functional request message
/MICROSAR/CanIf/CanIfInitCfg/CanIfRxPduCfg/CanIfRxPduCanId
- ▶ Configure the CAN ID type (standard or extended) and the DLC with the parameters
/MICROSAR/CanIf/CanIfInitCfg/CanIfRxPduCfg/CanIfRxPduCanIdType
/MICROSAR/CanIf/CanIfInitCfg/CanIfRxPduCfg/CanIfRxPduDlc
- ▶ Reference the related functional request Rx global Pdu created in the previous steps with the parameter
/MICROSAR/CanIf/CanIfInitCfg/CanIfRxPduCfg/CanIfRxPduRef
- ▶ Reference the related CAN channel hardware receive object (HRH) used for reception of the functional request message with the parameter
/MICROSAR/CanIf/CanIfInitCfg/CanIfRxPduCfg/CanIfRxPduHrhIdRef
- ▶ Assign the CanTp as upper layer user with the parameter /MICROSAR/CanIf/CanIfInitCfg/CanIfRxPduCfg/CanIfRxPduUserRxIndicationUL

For the example configuration this results in the following CanIf Rx Pdu configuration:

CanIf > CanIfInitCfg > CanIfRxPduCfgs > CanIfRxPdu_FuncReq_CAN0						
	Rx Pdu Can Id	Rx Pdu Can Id Type	Rx Pdu Dlc [Byte]	Rx Pdu Dlc Check	Rx Pdu Ref	Rx Pdu Hrh Id Ref
CanIfRxPdu_FuncReq_CAN0	0x7DF	STANDARD_CAN	8	<input checked="" type="checkbox"/>	RxPdu_FuncReq_CAN0	CN_Rx_CAN0



Create CanIf Tx Pdus

- ▶ Create a CanIf Tx Pdu for every destination channel where the functional requests shall be routed to:
CanIf Tx Pdu container: /MICROSAR/CanIf/CanIfInitCfg/CanIfTxPduCfg
- ▶ Configure the static Tx CAN ID of the functional request message with the parameter
/MICROSAR/CanIf/CanIfInitCfg/CanIfTxPduCfg/CanIfTxPduCanId
- ▶ Configure the CAN DLC with the parameter
/MICROSAR/CanIf/CanIfInitCfg/CanIfTxPduCfg/CanIfTxPduDlc
- ▶ Reference the related functional request Tx global Pdu created in the previous steps with the parameter
/MICROSAR/CanIf/CanIfInitCfg/CanIfTxPduCfg/CanIfTxPduRef
- ▶ Reference the related channel specific CanIf Tx buffer object used for transmission of the functional request message with the parameter
/MICROSAR/CanIf/CanIfInitCfg/CanIfTxPduCfg/CanIfTxPduBufferRef
- ▶ Assign the CanTp as upper layer user with the parameter /MICROSAR/CanIf/CanIfInitCfg/CanIfTxPduCfg/CanIfTxPduUserTxConfirmationUL

For the example configuration this results in the following CanIf Tx Pdu configuration:

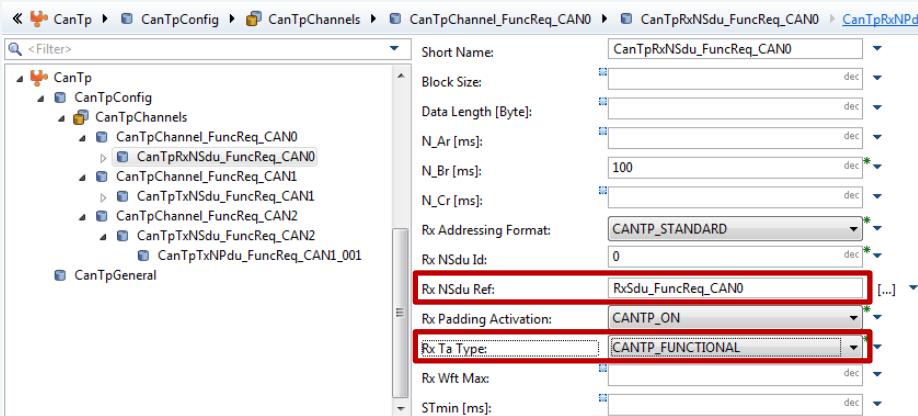
CanIf > CanIfInitCfg > CanIfTxPduCfgs > CanIfTxPdu_FuncReq_CAN1_CanIfTxPdu_FuncReq_CAN2						
	Tx Pdu Can Id	Tx Pdu Can Id Type	Tx Pdu Dlc [Byte]	Tx Pdu Buffer Ref	Tx Pdu Ref	Tx Pdu User Tx Confirmation UL
CanIfTxPdu_FuncReq_CAN1	0x7DF	STANDARD_CAN	8	CanIfBufferCfg_Tx_CAN1	TxPdu_FuncReq_CAN1	CAN_TP
CanIfTxPdu_FuncReq_CAN2	0x7DF	STANDARD_CAN	8	CanIfBufferCfg_Tx_CAN2	TxPdu_FuncReq_CAN2	CAN_TP



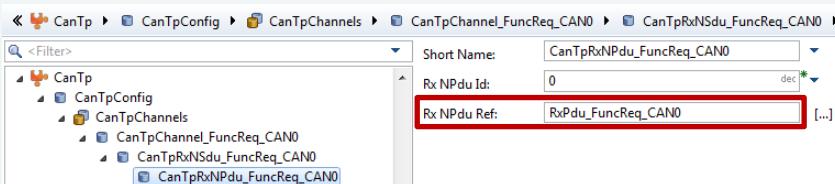
Create CanTp Rx channel

The following steps are optional if the own functional request routing path was automatically derived based on input files during project setup.

- ▶ Create a CanTp Rx channel for the functional request
CanTp Rx channel container: /MICROSAR/CanTp/CanTpConfig/CanTpChannel
- ▶ Create a CanTp Rx N-Sdu container below the previously created Rx channel:
/MICROSAR/CanTp/CanTpConfig/CanTpChannel/CanTpRxNSdu
- ▶ Reference the related global Pdu (Sdu, interconnecting the CanTp, PduR and diagnostic handler) created in the previous steps with the parameter
/MICROSAR/CanTp/CanTpConfig/CanTpChannel/CanTpRxNSdu/CanTpRxNSduRef
- ▶ Configure the Rx Sdu as functional communication type with the parameter
/MICROSAR/CanTp/CanTpConfig/CanTpChannel/CanTpRxNSdu/CanTpRxTaType



- ▶ Reference the related global Pdu (interconnecting the CanTp and CanTp) created in the previous steps with the parameter /MICROSAR/CanTp/CanTpConfig/CanTpChannel/CanTpRxNSdu/CanTpRxNPdu/CanTpRxNPduRef



For further information regarding the CanTp timing configuration parameters please refer to [7].

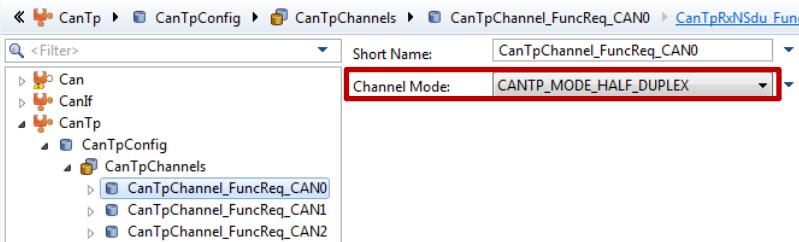


Create CanTp Tx channels

- ▶ Create a CanTp Tx channel for every destination channel where the functional requests shall be routed to.

CanTp Tx channel container: /MICROSAR/CanTp/CanTpConfig/CanTpChannel

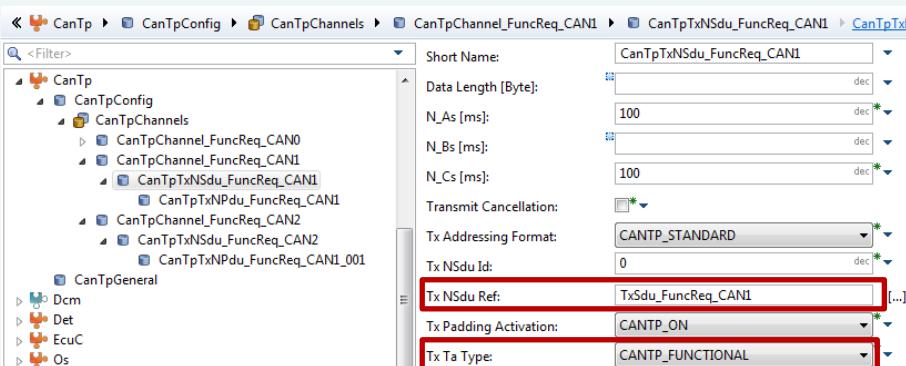
- ▶ Set the channel mode of the created Tx channel to half-duplex mode:
/MICROSAR/CanTp/CanTpConfig/CanTpChannel/CanTpChannelMode



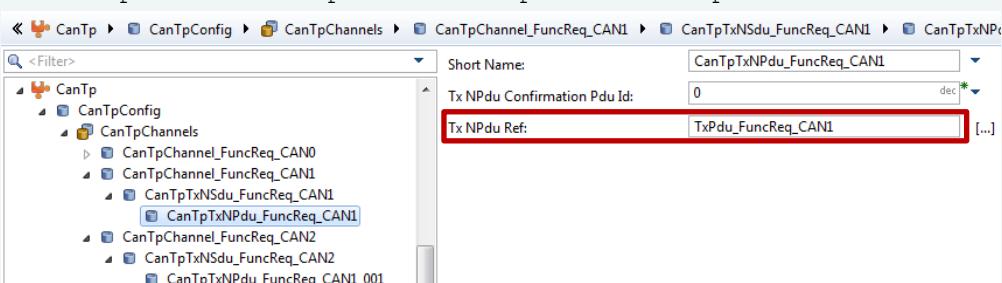
- ▶ Create a CanTp Tx N-Sdu container below the previously created Tx channel:
/MICROSAR/CanTp/CanTpConfig/CanTpChannel/CanTpTxNSdu

- ▶ Reference the related global Pdu (Sdu, interconnecting the CanTp, PduR and diagnostic handler) created in the previous steps with the parameter
/MICROSAR/CanTp/CanTpConfig/CanTpChannel/CanTpTxNSdu/CanTpTxNSduRef

- ▶ Configure the Tx Sdu as functional communication type with the parameter
/MICROSAR/CanTp/CanTpConfig/CanTpChannel/CanTpTxNSdu/CanTpTxTaType



- ▶ Reference the related global Pdu (interconnecting the Canlf and CanTp) created in the previous steps with the parameter /MICROSAR/CanTp/CanTpConfig/CanTpChannel/CanTpTxNSdu/CanTpTxNPdu/CanTpTxNPduRef



For further information regarding the CanTp timing configuration parameters please refer to [7].



Create a PduR Tx buffer

The following steps are optional if sufficient Tx buffers are already configured.

- ▶ Create a new PduR Tx buffer which will be used for the TP gateway routing of the functional request messages.
/MICROSAR/PduR/PduRRoutingTables/PduRTxBufferTable/PduRTxBuffer
- ▶ Configure the size (.../PduRTxBuffer/PduRPduMaxLength) of Tx buffer to the TP payload length of the functional requests. In case of CanTp the buffer size must be at least 7 bytes.

Please refer to chapter 2.6.3 for further details regarding the TP buffer configuration.



Create / Extend PduR routing path

Finally create a PduR 1:N routing path. This 1:N routing path shall route the received functional diagnostic requests to the gateway-own diagnostic application and all connected destination channels.

In case the functional request routing path for the gateway-own diagnostic application was derived automatically the first steps describing the creation of a routing path can be skipped. Proceed with the configuration of the additional routing destinations.

- ▶ Create the PduR routing path:

/MICROSAR/PduR/PduRRoutingTables/PduRRoutingTable/PduRRoutingPath

- ▶ Reference the related functional request Rx global Pdu (Sdu, interconnecting the CanTp, PduR and diagnostic handler) at the routing path source:

/MICROSAR/PduR/PduRRoutingTables/PduRRoutingTable/PduRRoutingPath/PduRSrcPdu/PduRSrcPduRef

and the routing path destination:

/MICROSAR/PduR/PduRRoutingTables/PduRRoutingTable/PduRRoutingPath/PduRDestPdu/PduRDestPduRef

This step is optional if the own functional request routing path was automatically derived based on input files during project setup.

- ▶ Add a new routing destination to the previously created routing path for every destination channel the functional requests shall be routed to

PduR routing path destination: /MICROSAR/PduR/

PduRRoutingTables/PduRRoutingTable/PduRRoutingPath/PduRDestPdu

and reference the related functional request Tx global Pdu (Sdu) at the new routing path destination:

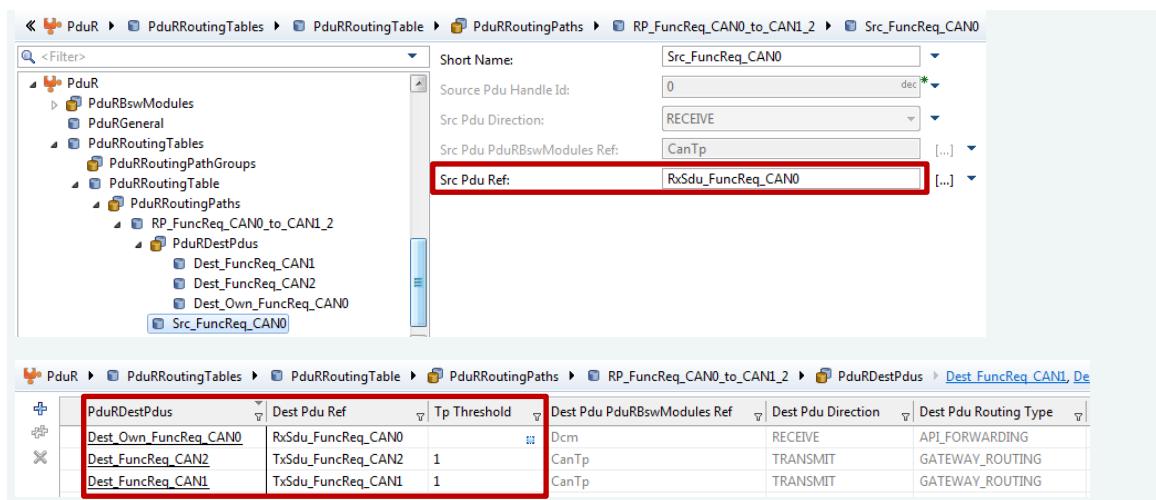
/MICROSAR/PduR/PduRRoutingTables/PduRRoutingTable/PduRRoutingPath/PduRDestPdu/PduRDestPduRef

- ▶ Configure the TP threshold value of the created gateway routing path to the value 1. Please refer to chapter 2.6.2 for further details of the TP threshold configuration.

/MICROSAR/PduR/PduRRoutingTables/PduRRoutingTable/PduRRoutingPath/PduRDestPdu/PduRTpThreshold

For the example configuration this results in the following PduR routing path configuration:

RoutingPath	RoutingPath Source Global Pdu	RoutingPath Destination(s) Global Pdu(s)
RP_FuncReq_CAN0_to_CAN1_2	RxSdu_FuncReq_CAN0	RxSdu_FuncReq_CAN0 (gateway-own diagnostic application), TxSdu_FuncReq_CAN1, TxSdu_FuncReq_CAN2



For further details about the CanIf and CanTp modules, please refer to [5] and [7].

5.3 Use Case Configuration: N:1 routing path

N:1 routing paths are not specified by AUTOSAR. The multiplicity of a PduRSrcPdu container is defined to one. Therefore, N single 1:1:1:M routing paths (mixture of 1:M/N:1 routing paths is supported) referencing the same global Pdu in one of their PduRDestPdu container must be configured. Every PduRSrcPdu container must reference their own global PDU, duplicates are not allowed. Whereas more than one PduRDestPdu container may reference the same global PDU to support the N:1 routing feature. An example configuration is shown in Figure 5-5. Both configured routing paths are 1:2 routing paths, but both routing paths refer to the same global PDU in one of their two PduRDestPdu container. That makes them also a 2:1 routing path. The data flow is shown in Figure 5-6. The names refer to the example configuration in Figure 5-5.



Cancel Transmit for N:1 routing paths is only supported if a Tx Confirmation is enabled.

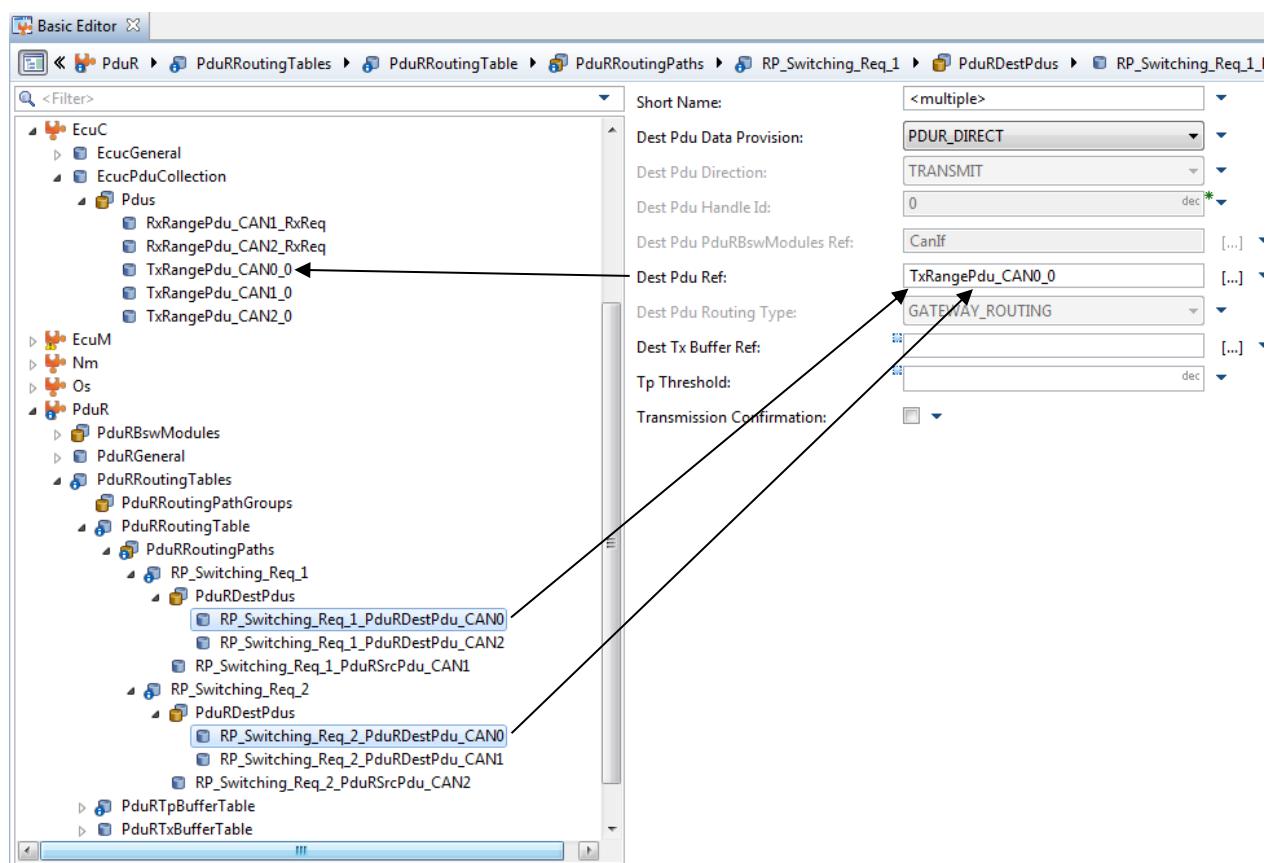


Figure 5-5 example N:1 routing path configuration

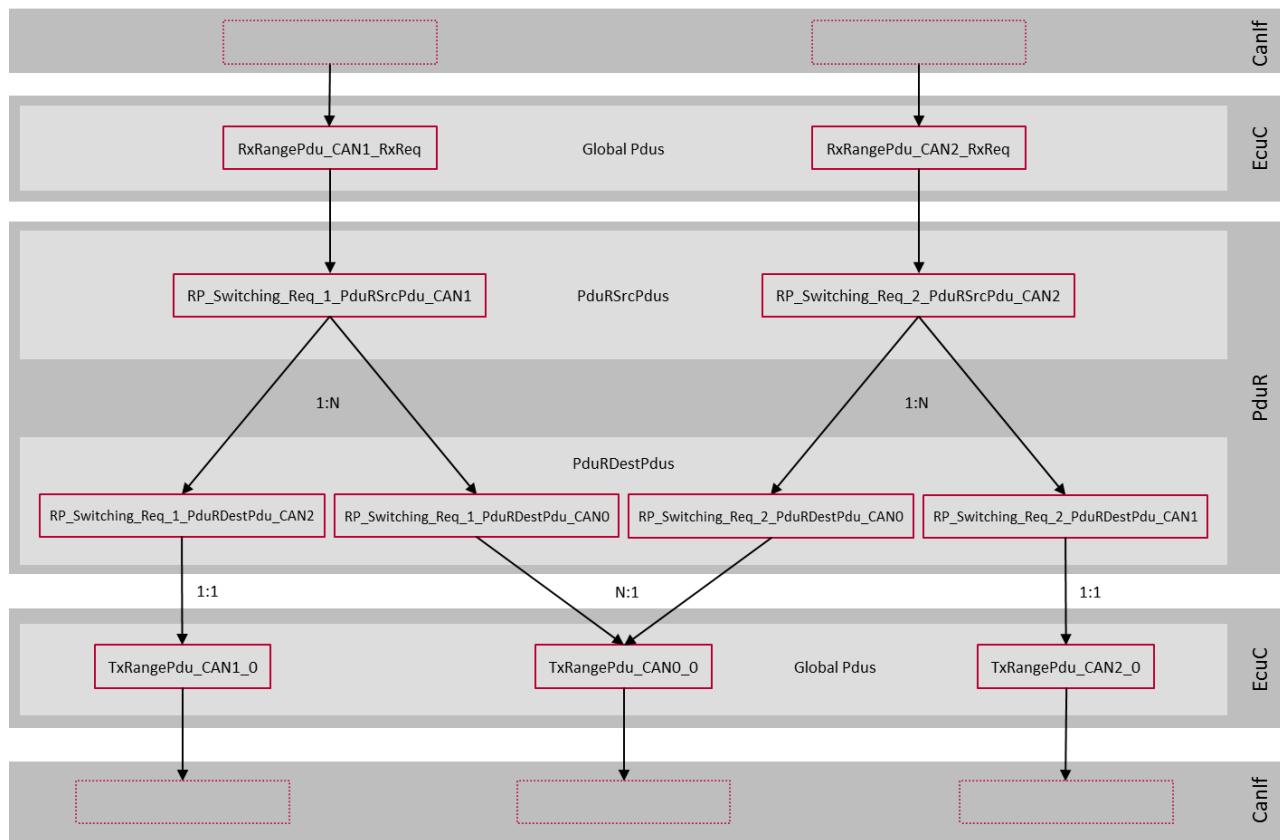


Figure 5-6 EcuC configuration of (mixed) N:1 / 1:N routing paths

6 Glossary and Abbreviations

6.1 Glossary

Term	Description
BSWMD	The BSWMD is a formal notation of all information belonging to a certain BSW artifact (BSW module or BSW cluster) in addition to the implementation of that artifact.
Buffer	A buffer in a memory area located in the RAM. It is a memory area reserved by the application for data storage.
Callback function	This is a function provided by an application. E.g. the CAN Driver calls a callback function to allow the application to control some action, to make decisions at runtime and to influence the work of the driver.
Cfg5	DaVinci Configurator
Channel	A channel defines the assignment (1:1) between a physical communication interface and a physical layer on which different modules are connected to (either CAN or LIN). 1 channel consists of 1..X network(s).
Component	CAN Driver, Network Management ... are software COMPONENTS in contrast to the expression module, which describes an ECU.
Confirmation	A service primitive defined in the ISO/OSI Reference Model (ISO 7498). With the service primitive 'confirmation' a service provider informs a service user about the result of a preceding service request of the service user. Notification by the CAN Driver on asynchronous successful transmission of a CAN message.
Critical section	A critical section is a sequence of instructions where mutual exclusion must be ensured. Such a section is called 'critical' because shared data is modified within it.
Electronic Control Unit	Also known as ECU. Small embedded computer system consisting of at least one CPU and corresponding periphery which is placed in one housing.
Event	An exclusive signal which is assigned to a certain extended task. An event can be used to send binary information to an extended task. The meaning of events is defined by the application. Any task can set an event for an extended task. The event can only be cleared by the task which is assigned to the event.
Gateway	A gateway is designed to enable communication between different bus systems, e.g. from CAN to LIN.
Indication	A service primitive defined in the ISO/OSI Reference Model (ISO 7498). With the service primitive 'indication' a service provider informs a service user about the occurrence of either an internal event or a service request issued by another service user. Notification of application in case of events in the Vector software components, e.g. an asynchronous reception of a CAN message.
Interrupt	Processor-specific event which can interrupt the execution of a current program section.

Interrupt service routine	The function used for direct processing of an interrupt.
Network	A network defines the assignment (1:N) between a logical communication grouping and a physical layer on which different modules are connected to (either CAN or LIN). One network consists of one channel, Y networks consists of 1..Z channel(s). We say network if we talk about more than one bus.
Overrun	Overwriting data in a memory with limited capacity, e.g. queued message object.
Post-build	This type of configuration is possible after building the software module or the ECU software. The software may either receive parameters of its configuration during the download of the complete ECU software resulting from the linkage of the code, or it may receive its configuration file that can be downloaded to the ECU separately, avoiding a re-compilation and re-build of the ECU software modules. In order to make the post-build time re-configuration possible, the re-configurable parameters shall be stored at a known memory location of ECU storage area.
Schedule table	Table containing the sequence of LIN message identifiers to be transmitted together with the message delay.
Signal	A signal is responsible for the logical transmission and reception of information depending on the restrictions of the physical layer. The definition of the signal contents is part of the database given by the vehicle manufacturer. Signals describe the significance of the individual data segments within a message. Typically bits, bytes or words are used for data segments but individual bit combinations are also possible. In the CAN data base, each data segment is assigned a symbolic name, a value range, a conversion formula and a physical unit, as well as a list of receiving nodes.
Transport Protocol	Some information that must be transferred over the CAN/LIN bus does not fit into individual message frames because the data length exceeds the maximum of 8 bytes. In this case, the sender must divide up the data into a number of messages. Additional information is necessary for the receiver to put the data together again.
Use case	A model of the usage by the user of a system in order to realize a single functional feature of the system.

Table 6-1 Glossary

6.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
BSWMD	Basic Software Module Description
CAN	Controller Area Network protocol originally defined for use as a communication network for control applications in vehicles.
CDD	Complex Device Driver

COM	Communication
DCM	Diagnostic Communication Manager
DEM	Diagnostic Event Manager
DET	Development Error Tracer
DLC	Data Length Code, Number of data bytes of a CAN message
EAD	Embedded Architecture Designer
ECU	Electronic Control Unit
ECUC	ECU Configuration
FIFO	First In First Out
HIS	Hersteller Initiative Software
ID	Identifier (e.g. Identifier of a CAN message)
ISR	Interrupt Service Routine
LIN	Local Interconnect Network
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
MSN	Module shortname
PDU	Protocol Data Unit
PDUR	PDU Router
RAM	Random Access Memory
ROM	Read-Only Memory
RTE	Runtime Environment
SDU	Segmented Data Unit
SRS	Software Requirement Specification
SWC	Software Component
SWS	Software Specification
TP	Transport Protocol

Table 6-2 Abbreviations

7 Contact

Visit our website for more information on

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

www.vector.com