MICROSAR PDU Router

Technical Reference

DaVinci Configurator

Version 7.00.00

Authors	Erich Schondelmaier, Gunnar Meiss, Sebastian Waldvogel, Florian Röhm, Büsra Bayrak, Heiko Hübler, Michal Al-Jazani
Status	Released

Document Information

History

Author	Date	Version	Remarks
Erich Schondelmaier	2012-12-20	1.00.00	Initial version based on PduR Technical Reference
Erich Schondelmaier	2012-07-12	2.00.00	Adapted to AUTOSAR 4.0.3
Erich Schondelmaier	2012-10-15	2.01.00	TP Gateway IF Gateway
Gunnar Meiss	2012-11-21	2.02.00	AR4-285: Support PduRRoutingPathGroups
Erich Schondelmaier	2013-02-07	2.02.01	Adapted Tp- API description
Erich Schondelmaier	2013-02-15	2.02.02	Added some ASR deviations ESCAN00064126
Erich Schondelmaier	2013-03-19	2.03.00	ESCAN00064364 AR4-325: Post-Build Loadable Added Cancel- Receive/ Transmit Support
Erich Schondelmaier	2014-04-15	2.04.00	Added TP routing with variable addresses (MetaData Handling) Added Threshold “0” support
Erich Schondelmaier	2014-04-15	2.04.01	Support the StartOfReception API (with the PduInfoType), TxConfirmation and RxIndication according ASR4.1.2
Erich Schondelmaier	2014-09-01	2.05.00	Added SecOC to the Interface Overview Extended Tp Gateway Routing behavior description Updated Configuration Variant
Sebastian Waldvogel	2015-02-23	2.06.00	FEAT-1057: Added documentation about configuration of range routing paths and functional requests gateway
Sebastian Waldvogel	2015-05-11	2.06.01	FEAT-1057: Improvements of documentation
Florian Röhm	2015-07-30	2.07.00	FEAT-109: Added documentation for PduR switching feature and N:1 routing paths
Florian Röhm	2016-01-16	2.08.00	FEAT-1485: Added documentation for 1:N and N:1 transport protocol routing paths
Gunnar Meiss	2016-02-25	2.08.00	FEAT-1631: Trigger Transmit API with SduLength In/Out according to ASR4.2.2
Erich Schondelmaier	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 suppport only for lower layer CanIf.
Florian Röhm	2016-04-01	2.08.01	Removed empty chapters
Erich Schondelmaier, Florian Röhm	2016-08-10	3.00.00	Shared/Dedicated Buffer support Memory mapping extension
Sebastian Waldvogel	2016-11-24	3.00.00	Smart Learning (Switching)
Florian Röhm	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
Florian Röhm	2017-06-23	3.00.01	STORYC-1629: N:1 routing path support for IpduM Container feature
Florian Röhm	2017-09-21	3.01.00	STORYC-1972: Support 1:N Tx IF API Forwarding
Büsra Bayrak	2017-11-07	3.02.00	STORYC-2031: Support buffered IF API Forwarding routing paths
Florian Röhm	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
Heiko Hübler	2018-03-07	3.04.00	Maintenance (STORYC-3934)
Heiko Hübler	2018-03-14	4.00.00	STORYC-3241: Implement Multicore Support
Florian Röhm	2018-07-04	4.01.00	STORYC-4890: Release PduRBswModules on different Cores
Florian Röhm	2019-02-25	5.00.00	STORYC-7435: Release [GW] Message Buffering in MICROSAR
Florian Röhm	2019-04-17	5.01.00	STORYC-7752: On-the-fly routing of (J1939) TP messages
Florian Röhm	2019-08-12	5.02.00	ESCAN00103997:Old term 'TpBuffer' is still used instead of 'TxBuffer'
Florian Röhm	2019-10-17	6.00.00	MSR4-117: PduR Support for Mixed ASIL Systems
Florian Röhm	2019-11-21	6.01.00	COM-1247: Improve documentation for Mixed ASIL MultiPartition Usecase
Michal Al-Jazani	2020-02-27	6.02.00	COM-1409: PduR_PreInit documentation
Florian Röhm	2020-07-07	7.00.00	COM-1300: Provide MC Queue Configuration Per Queue

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_SWS_PDURouter.pdf	4.0.3
[2]	AUTOSAR	AUTOSAR_SWS_PDURouter.pdf.	4.1.1
[3]	AUTOSAR	AUTOSAR_SWS_PDURouter.pdf	4.1.2
[4]	AUTOSAR	AUTOSAR_SWS_DevelopmentErrorTracer.pdf	3.2.0
[5]	AUTOSAR	AUTOSAR_TR_BSWModuleList.pdf	1.6.0
[6]	AUTOSAR	AUTOSAR_SWS_SAEJ1939TransportLayer.pdf	1.5.0
[7]	Vector	TechnicalReference_CanIf.pdf	6.02.00
[8]	Vector	TechnicalReference_<CAN Driver>.pdf	-
[9]	AUTOSAR	TechnicalReference_CanTp.pdf	2.00.00

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

Illustrations

Tables

1 Component History

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

Component Version	New Features
1.00	▶ MICROSAR 4 Base
2.00	▶ AUTOSAR 4.0.3
2.01	▶ TP Gateway ▶ IF Gateway
2.02	▶ Routing Path Groups
2.03	▶ Post-Build Loadable
5.00	▶ Changed StartOfReception, TpRxIndication and TpTxConfirmation APIs according to AUTOSAR 4.1.2 ▶ Added TP routing with variable addresses ▶ Meta-Data support
6.00	▶ Post-Build Selectable
7.00	▶ CAN-FD
8.00	▶ PduR Switching ▶ N:1 Interface routing path support
9.00	▶ 1:N/N:1 transport protocol routing path support
11.00	▶ Tx Queue overflow notification
13.00	▶ Support Tx If Pdu Fan-out
14.02	▶ Support Multicore
15.03	▶ Support optimized If queue; Support queue configuration container
15.04	▶ Support on-the-fly routing of 1:N Tp routing with UL destination
16.00	▶ Cleanup component
17.00	▶ PduR Support for Mixed ASIL Systems

Table 1-1 Component history

2 Introduction

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

Supported AUTOSAR Release*:	4	
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. [5])

* For the precise AUTOSAR Release 4.x please see the release specific documentation.

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.

2.1 Architecture Overview

Figure 2-1 shows where the PDUR is located in the AUTOSAR architecture.

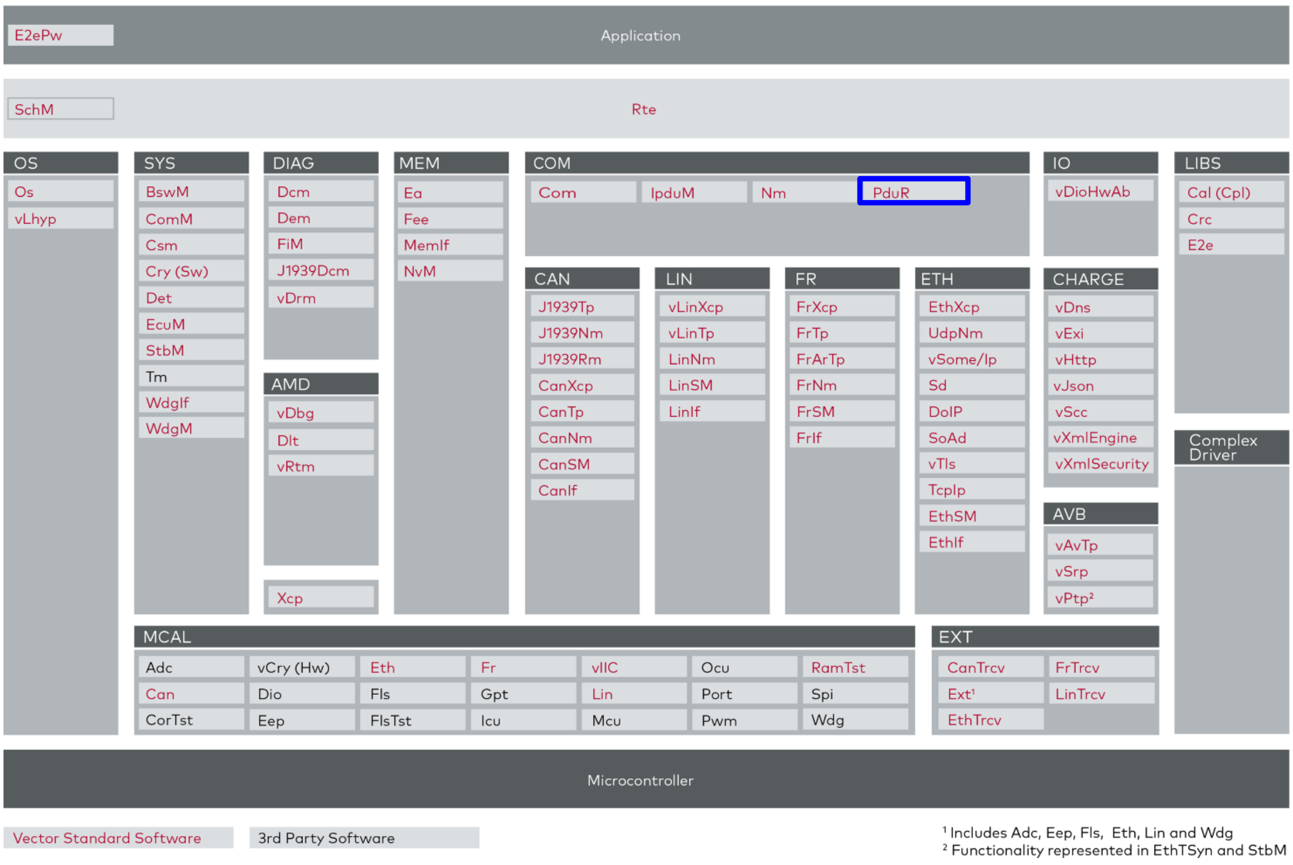


Figure 2-1 AUTOSAR 4.1 Architecture Overview

Figure 2-2 shows the interfaces to adjacent modules of the PDUR. These interfaces are described in chapter 0.

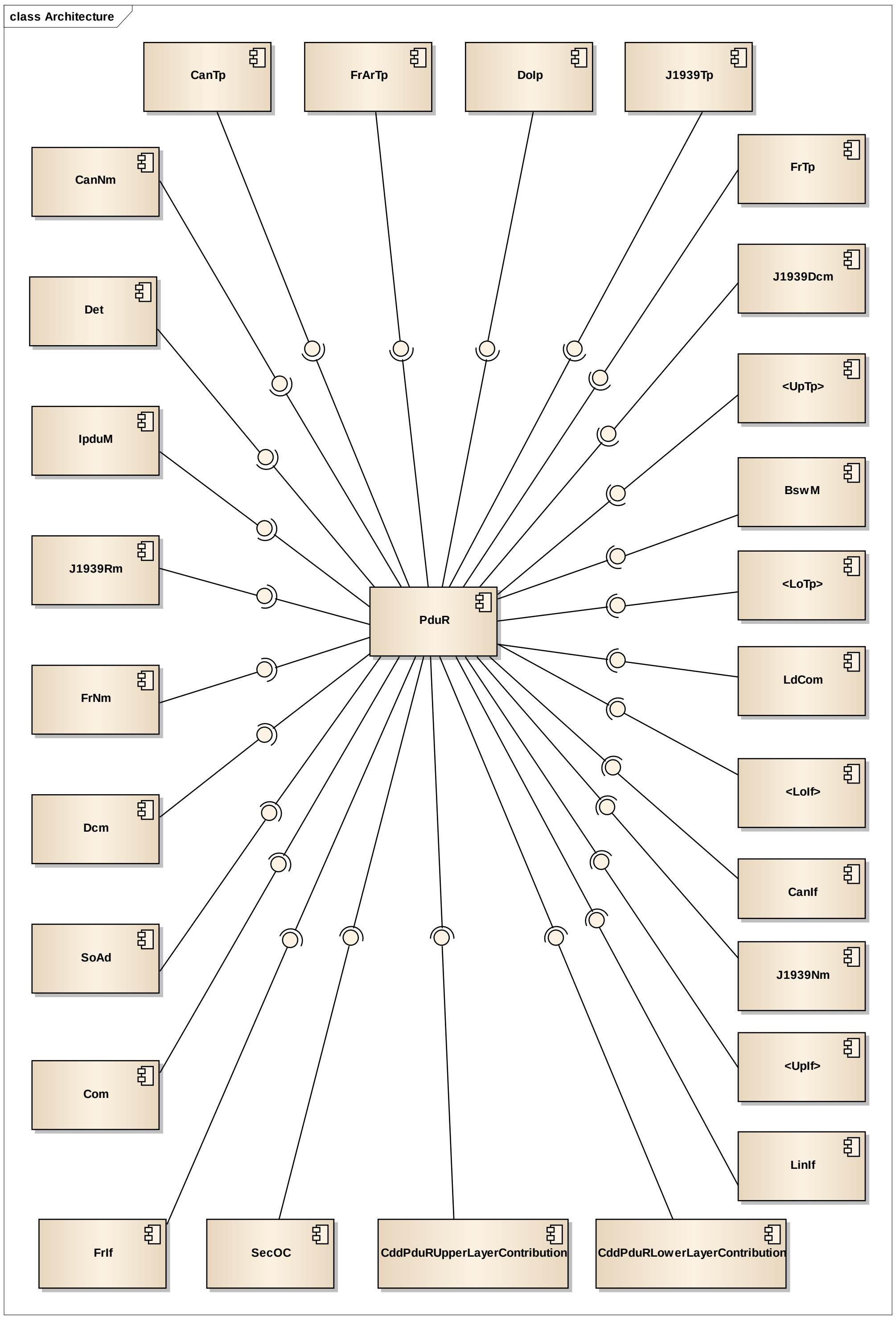


Figure 2-2 Interfaces to adjacent modules of the PDUR

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 5 and are defined in [1].

3 Functional Description

3.1 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 3-1 Supported AUTOSAR standard conform features
- Table 3-2 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 3-3 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 3-1 Supported AUTOSAR standard conform features

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

Not Supported AUTOSAR Standard Conform Features
Link time configuration
1:N fan-out from the same upper layer PDU (Tp)
Zero cost operation
Minimum Routing (Reduced state)

Table 3-2 Not supported AUTOSAR standard conform features

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)
PduR Switching: Smart learning of routing path destinations depending on received Pdus
1:N and N:1 transport protocol single- and multiframe routing path capability
Choose between different Queues/Queue Implementations

Table 3-3 Features provided beyond the AUTOSAR standard

3.2 Interfaces to adjacent modules of the PDUR

The PDU Router provides generic communication interfaces and transport protocol APIs for any lower and upper layer module.

3.3 Initialization

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

PduR_PreInit has to be called only once by the EcuMDriverInitListOne before the start of the OS. This applies also to a multi OsApplication/Core system.

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

Note
In case of a single OsApplication/Core system PduR_Init must be called directly after PduR_PreInit.

3.4 States

The PduR is initially not activated. Basic RAM arrays are initialized with the call of PduR_InitMemory or with the startup code of your ECU.

PduR_PreInit initializes all shared memory used by the PduR. Afterwards it is in the PreInitialized state.

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

3.5 Error Handling

3.5.1 Development Error Reporting

By default, development errors are reported to the DET using the service `Det_ReportError()` as specified in [4], 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_GROUP_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 OsApplication context.
PDUR_E_SPINLOCKTIMEOUT	The PduR ran into a timeout while trying to acquire a spinlock.
PDUR_E_IDXOUTOFBOUNDS	A PduR variable was read which had an invalid value.
PDUR_E_OSAPP2MEMMAPPING	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.

Table 3-4 Errors reported to DET

3.6 Communication Interface Routing

3.6.1 Data Provision

3.6.1.1 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.

3.6.1.2 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.

3.6.2 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 29): FIFO behavior
- Shared Buffer Queue (page 30): FIFO behavior
- Single Buffer Queue (page 29): Last-is-best behavior

3.6.2.1 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)

3.6.2.2 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.

3.6.3 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.

3.6.4 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.

3.6.5 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.

3.6.6 Smart Learning (Switching)

The PduR routing path relations are statically configured and cannot be modified during runtime. In case ECUs will be attached to different networks during assembly or a huge amount of possible routing relations will be required, the dynamic learning of routing relations can be used.

The switching functionality is based on a dynamic RAM table, the forwarding information base (FIB) inside the PduR. This RAM table stores the location (network) of the different ECUs and will be used to direct the routings to the correct destination. The identification of the communication partners is done by a source- and target address which is determined based on the PDU meta data. The FIB is updated with the related location (called connection) on reception of every participating PDU.

Source address	Learned destination location/connection
0x00	Connection 2
0x01	<not yet learned>
0x02	Connection 0
…	…

Table 3-5 Example FIB content after reception of Pdus from source 0x00 and 0x02

On reception of every PDU a lookup in the FIB is done to check whether the location/connection of the target address is already known. If the target address destination (FIB source address) is not yet known, the PDU will be broadcasted to all destinations of the respective routing path. In case the target address destination was already learned by the gateway, the PDU will be routed to the learned destination instead of the broadcasting.

From the technical point of view the switching functionality is an add-on for the static PduR routing path relations. The standard PduR routing paths are used to describe all required routing relations which are necessary if no dynamic learning is available. This means that the routing paths must be configured in a way that the PDUs are broadcasted to all desired networks which have possible destination ECUs connected. This is typically a 1:N routing path which performs a broadcasting of the PDU. The switching add-on suppresses the routing to the individual destinations based on the FIB information. In case the destination of a single PDU / target address is not yet known, the PDU will be routed to all destinations of the 1:N routing path. In case the destination is already learned, the routing to all destinations except the learned one is suppressed.

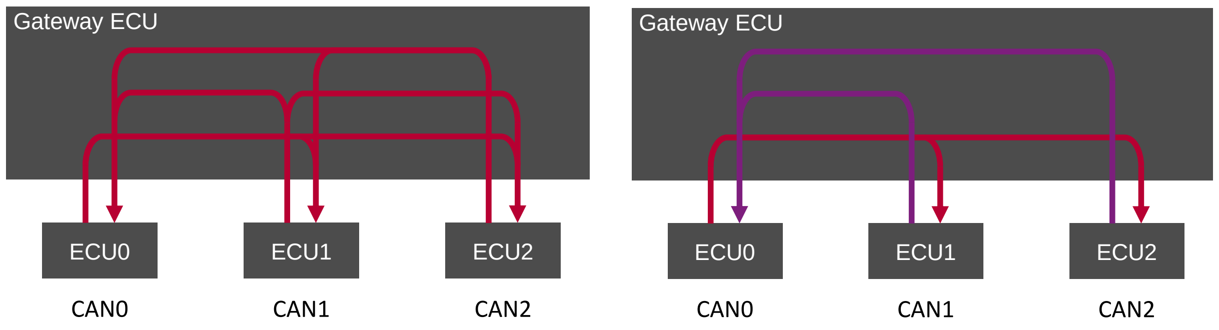


Figure 3-1 Example routing path configurations: Every ECU transmits to every other ECU (via N:M routing paths) or ECU0 can exclusively broadcast to all other ECUs. Other ECUs can only reach ECU0

An important concept for the smart learning functionality is the usage of communication interface range routing paths where multiple PDUs can be routed with a single routing path. For the examples this means that only three N:M or 1:N/N:1 routing paths have to be configured between the networks. The range filters of the Rx CAN range Pdus can be used to define the concrete CAN ID range to be routed via the routing paths. For further information see chapter 6.1.

The memorization of the source / target address locations/networks is done on the basis of the actual extracted source / and target addresses which are based on the CAN ID of the routed range PDUs.

Caution
The PduR switching feature is only supported for MetaData Pdus. Therefore the referenced routing paths in a PduR Connection must be MetaData routing paths. Currently only the CanIf supports the MetaData feature.

The switching functionality is automatically enabled for all PDUs associated to a `PduRDataPlane` (see chapter 3.6.6.1).

All entries of the FIB will be set to 'not yet learned' during initialization of the PduR. This is the only way to completely 'unlearn' the FIB table. A relearning is possible by receiving messages with the corresponding source address on some other channel. The PduR will then update the connection/location of this address.

3.6.6.1 Configuration

The configuration and activation of the switching functionality is done within the EcuC container `/MICROSAR/PduR/PduRRoutingTables/PduRDataPlaneTable`. By adding a new `PduRDataPlane` an independent FIB RAM table will be instantiated. Therefore it is possible to configure multiple different independent smart learning / switching behaviors in a single gateway.

Within a `PduRDataPlane` the `PduRDataPlane/PduRConnection` containers are used to assign the static PduR routing path sources and destinations to a network (location). A `PduRConnection` represents a single network.

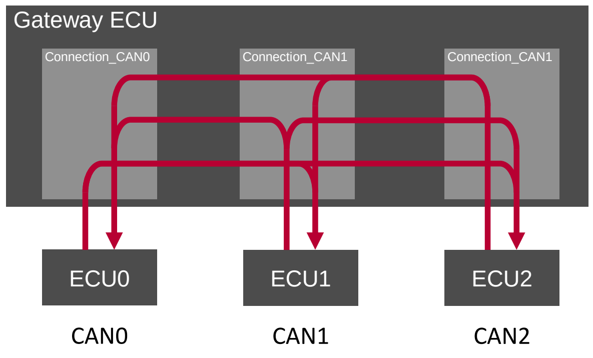


Figure 3-2 Network specific assignment of PduRRouting path sources and destinations to a PduRConnection

Reference all related `PduRRoutingPath/PduRSrcPdu` and `PduRRoutingPath/PduRDestPdu` of a single network to the `PduRConnection` by the references `PduRConnection/PduRSrcPduRef` and `PduRConnection/PduRDestPduRef`.

The behavior of the switching logic itself is configured in the `PduRDataPlane/PduRSwitchingStrategy/PduRSaTaSwitchingStrategy` container. The currently available strategy is based on source- and target addresses as described above.

Within the `PduRSaTaSwitchingStrategy` different strategies are supported to determine the source- and target address of the PDUs:

- **PduRLinearTaToSaCalculationStrategy**

  The target address is the CAN ID (contained in the Pdu meta data). The source address is calculated by masking the target address and adding a linear offset.

  For PDUs referenced by `PduRAddOffsetSrcPduRef`:
  ```
      source address = (target address + offset) & mask
  ```

For PDUs referenced by `PduRSubtractOffsetSrcPduRef`:

```
    source address = (target address - offset) & mask
```

- **PduRSaTaFromMetaDataCalculationStrategy**

  The source and target address is directly read from the CAN ID contained in the meta data. The bit access will occur on the logical CAN ID extracted from the meta data.

  The source address bit position is defined by the `PduRSaStartBitPosition` and `PduRSaEndBitPosition` parameters. The target address bit position is defined by the `PduRTaStartBitPosition` and `PduRTaEndBitPosition` parameters.

  Example bit position configuration of the CAN ID layout shown in Figure 3-3:

```
PduRSaStartBitPosition: 0
PduRSaEndBitPosition:   9
PduRTaStartBitPosition: 10
PduRTaEndBitPosition:   19
```

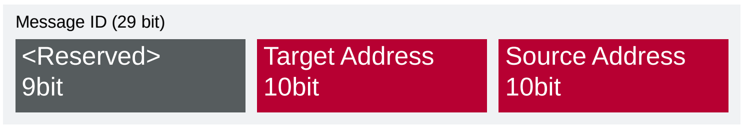


Figure 3-3 Example Extended CAN ID layout

EcuC structural changes with PduR version 9.00.00

PduRConnection

In the previous versions the `PduRConnection/PduRSrcPduRef` and `PduRConnection/PduRDestPduRef` parameters were used to separate between request and response routing paths. A separation between request and response routing paths does not exist anymore. Instead the references are now just used for network to `PduRSrcPdu` / `PduRDestPdu` mapping.

An automated migration of the `PduRConnection` references is not possible. Please add missing references manually.

PduRLinearMappingStrategy

In the previous versions the mapping between request and response PDUs was configured in the `PduRDataPlane/PduRDataPlaneMappingStrategy/PduRLinearMappingStrategy` container. This container will be automatically migrated to the new location `PduRDataPlane/PduRSwitchingStrategy/PduRSaTaSwitchingStrategy/PduRSaTaCalculationStrategy/PduRLinearTaToSaCalculationStrategy`. As described above the separation between request and response routing paths by the `PduRConnection/PduRSrcPduRef` and `PduRConnection/PduRDestPduRef` does not exist anymore. Rather the new references `PduRLinearTaToSaCalculationStrategy/PduRAddOffsetSrcPduRef` and `PduRLinearTaToSaCalculationStrategy/PduRSubtractOffsetSrcPduRef` were introduced. `PduRAddOffsetSrcPduRef` must reference all `PduRSrcPdus` where the offset shall be added to the target address to determine the related source address. PDUs where the offset shall be subtracted must be referenced with the `PduRSubtractOffsetSrcPduRef` references.

The `AddOffset-` and `SubtractOffsetSrcPduRef` references are automatically migrated. Former `PduRSrcPdu` referenced by `PduRConnection/PduRDestPduRef` are migrated to `PduRAddOffsetSrcPduRef`. Former `PduRSrcPdu` referenced by `PduRConnection/PduRSrcPduRef` are migrated to `PduRSubtractOffsetSrcPduRef`.

3.6.6.2 Example

Network topology is as shown in Figure 3-4. The CAN ID layout is as shown in Figure 3-5. Strategy `PduRSaTaFromMetaDataCalculationStrategy` used (see chapter 3.6.6.1). Routing path configuration as shown in Figure 3-1 (every ECU can broadcast to the other ECUs).

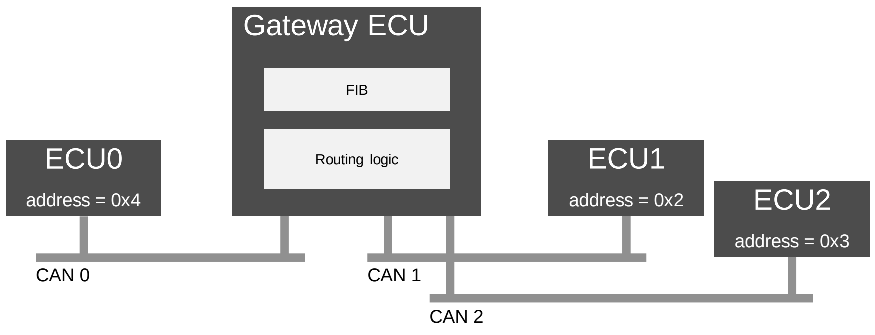


Figure 3-4 Example switching network topology

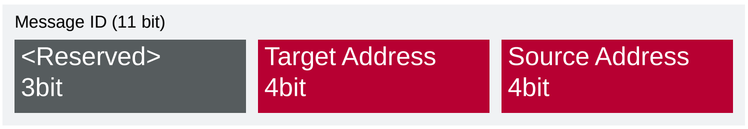
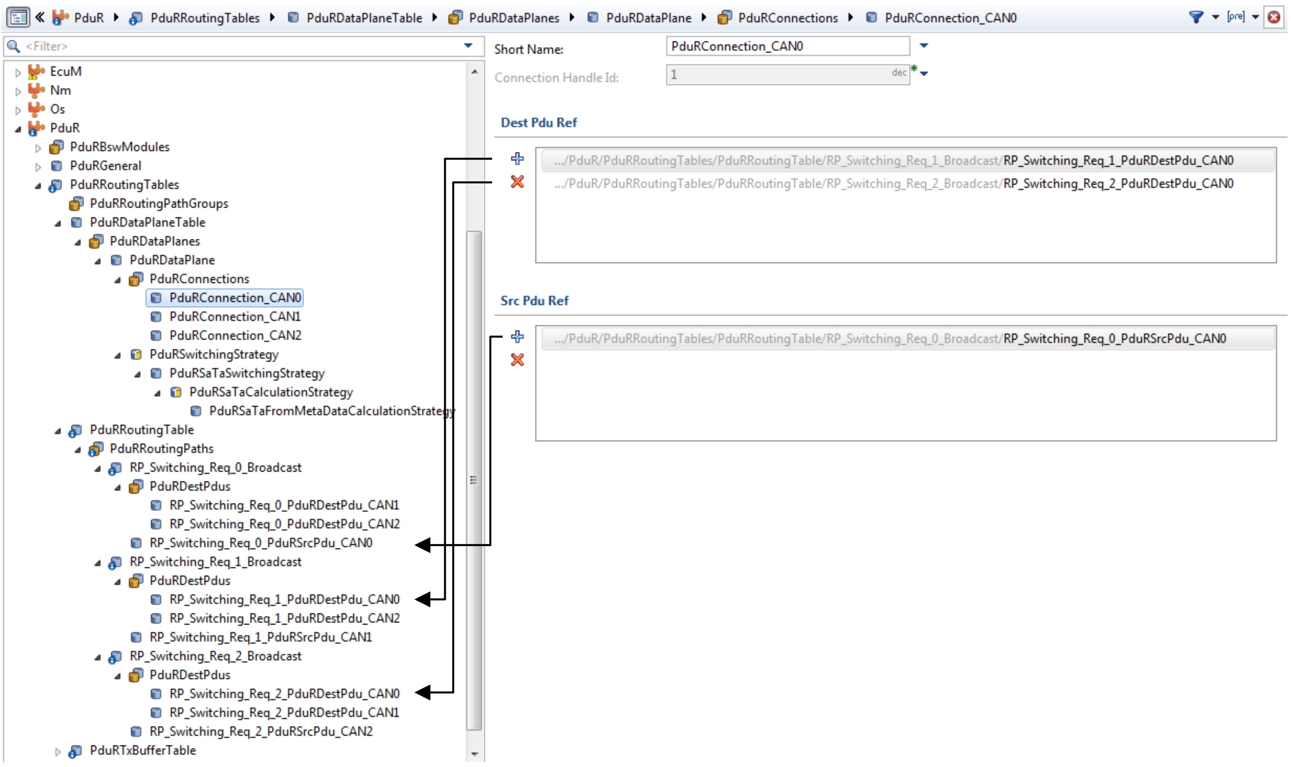
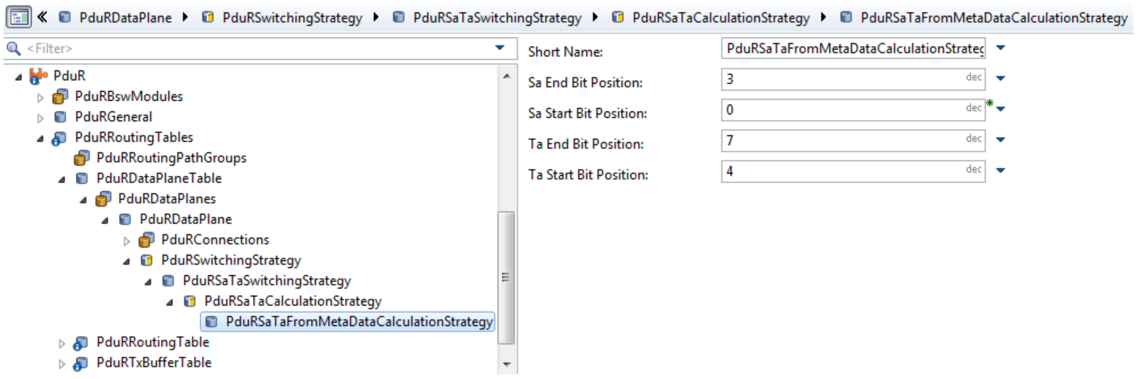


Figure 3-5 Example Standard CAN ID layout



```
« PduR ▸ PduRRoutingTables ▸ PduRDataPlaneTable ▸ PduRDataPlanes ▸ PduRDataPlane ▸ PduRConnections ▸ PduRConnection_CAN0

<Filter>
▷ EcuM
▷ Nm
▷ Os
◢ PduR
    ▷ PduRBswModules
    ▷ PduRGeneral
    ◢ PduRRoutingTables
          PduRRoutingPathGroups
        ◢ PduRDataPlaneTable
            ◢ PduRDataPlanes
                ◢ PduRDataPlane
                    ◢ PduRConnections
                          PduRConnection_CAN0
                          PduRConnection_CAN1
                          PduRConnection_CAN2
                    ◢ PduRSwitchingStrategy
                        ◢ PduRSaTaSwitchingStrategy
                            ◢ PduRSaTaCalculationStrategy
                                  PduRSaTaFromMetaDataCalculationStrategy
        ◢ PduRRoutingTable
            ◢ PduRRoutingPaths
                ◢ RP_Switching_Req_0_Broadcast
                    ◢ PduRDestPdus
                          RP_Switching_Req_0_PduRDestPdu_CAN1
                          RP_Switching_Req_0_PduRDestPdu_CAN2
                      RP_Switching_Req_0_PduRSrcPdu_CAN0
                ◢ RP_Switching_Req_1_Broadcast
                    ◢ PduRDestPdus
                          RP_Switching_Req_1_PduRDestPdu_CAN0
                          RP_Switching_Req_1_PduRDestPdu_CAN2
                      RP_Switching_Req_1_PduRSrcPdu_CAN1
                ◢ RP_Switching_Req_2_Broadcast
                    ◢ PduRDestPdus
                          RP_Switching_Req_2_PduRDestPdu_CAN0
                          RP_Switching_Req_2_PduRDestPdu_CAN1
                      RP_Switching_Req_2_PduRSrcPdu_CAN2
        ▷ PduRTxBufferTable

Short Name:               PduRConnection_CAN0
Connection Handle Id:     1   dec

Dest Pdu Ref
  .../PduR/PduRRoutingTables/PduRRoutingTable/RP_Switching_Req_1_Broadcast/RP_Switching_Req_1_PduRDestPdu_CAN0
  .../PduR/PduRRoutingTables/PduRRoutingTable/RP_Switching_Req_2_Broadcast/RP_Switching_Req_2_PduRDestPdu_CAN0

Src Pdu Ref
  .../PduR/PduRRoutingTables/PduRRoutingTable/RP_Switching_Req_0_Broadcast/RP_Switching_Req_0_PduRSrcPdu_CAN0
```

Figure 3-6 Example switching EcuC configuration - PduRConnection



```
« PduRDataPlane ▸ PduRSwitchingStrategy ▸ PduRSaTaSwitchingStrategy ▸ PduRSaTaCalculationStrategy ▸ PduRSaTaFromMetaDataCalculationStrategy

<Filter>
◢ PduR
    ▷ PduRBswModules
    ▷ PduRGeneral
    ◢ PduRRoutingTables
          PduRRoutingPathGroups
        ◢ PduRDataPlaneTable
            ◢ PduRDataPlanes
                ◢ PduRDataPlane
                    ▷ PduRConnections
                    ◢ PduRSwitchingStrategy
                        ◢ PduRSaTaSwitchingStrategy
                            ◢ PduRSaTaCalculationStrategy
                                  PduRSaTaFromMetaDataCalculationStrategy
        ▷ PduRRoutingTable
        ▷ PduRTxBufferTable

Short Name:               PduRSaTaFromMetaDataCalculationStrateg
Sa End Bit Position:      3   dec
Sa Start Bit Position:    0   dec
Ta End Bit Position:      7   dec
Ta Start Bit Position:    4   dec
```

Figure 3-7 Example switching EcuC configuration - PduRSaTaFromMetaDataCalculationStrategy

Example communication sequence and FIB contents:

1. ECU0 transmits the range Pdu with CAN ID **0x034**. This means ECU0 uses its address as source

Source Address	Learned location
…	…

address (0x4), and addresses ECU2 with the target address 0x3. The PDU will be broadcasted to CAN1 and CAN2 and the location of ECO0 address is updated in the FIB.

0x02	<not yet learned>
0x03	<not yet learned>
0x04	**Connection_CAN0**
…	…

2. ECU2 responds to the initial PDU of ECU0 with a PDU with CAN ID **0x043**. The PDU will only be routed to CAN0 because the location of the target address (0x4) was already learned by the first PDU of ECU0. Additionally the location of ECU2 is updated in the FIB.

Source Address	Learned location
…	…
0x02	<not yet learned>
0x03	**Connection_CAN2**
0x04	Connection_CAN0
…	…

3. Finally ECU1 transmits a PDU with CAN ID **0x032**. The PDU will only be routed to CAN2 because the location of the target address (0x3) was already learned. The FIB gets updated with the location of ECU1.

Source Address	Learned location
…	…
0x02	**Connection_CAN1**
0x03	Connection_CAN2
0x04	Connection_CAN0
…	…

3.6.7 Queue overflow notification callback

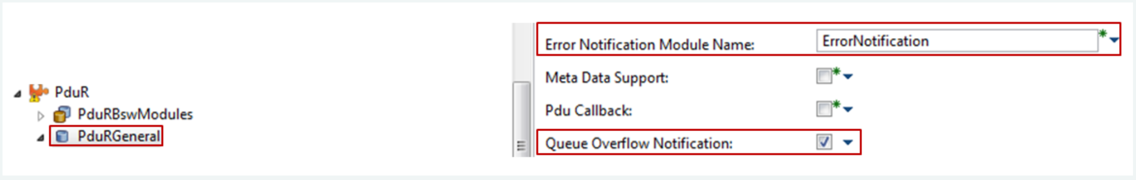
The PDU Router supports a Queue overflow notification callback. In case of a communication interface routing with unfavorable FIFO configuration or if the destination bus is not available (e.g. bus off) a buffer overflow can occur. In this case the FIFO is flushed.

Additionally, a callback could be configured to capture this event and perform error handling. See Figure 3-8.

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.



Error Notification Module Name: ErrorNotification

Meta Data Support: ☐

Pdu Callback: ☐

Queue Overflow Notification: ☑

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.

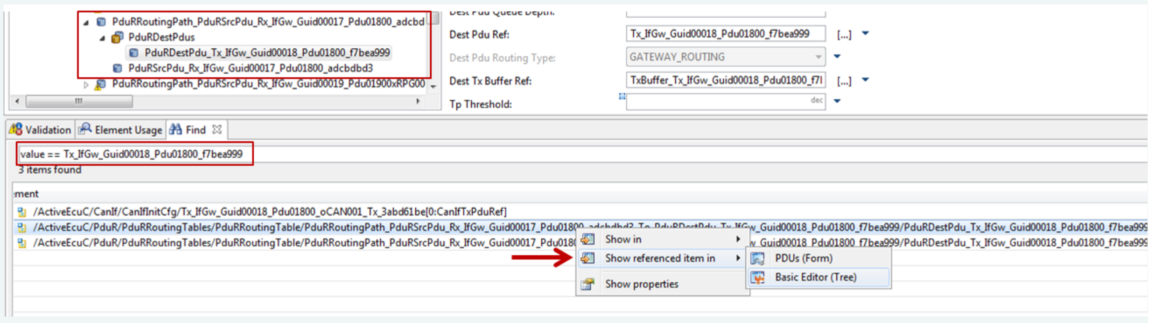
Figure 3-8 Overflow notification callback configuration

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.


PduRRoutingPath_PduRSrcPdu_Rx_IfGw_Guid00017_Pdu01800_adcbd
PduRDestPdus
PduRDestPdu_Tx_IfGw_Guid00018_Pdu01800_f7bea999
PduRSrcPdu_Rx_IfGw_Guid00017_Pdu01800_adcbdbd3
PduRRoutingPath_PduRSrcPdu_Rx_IfGw_Guid00019_Pdu01900xRPG00

Dest Pdu Ref: Tx_IfGw_Guid00018_Pdu01800_f7bea999
Dest Pdu Routing Type: GATEWAY_ROUTING
Dest Tx Buffer Ref: TxBuffer_Tx_IfGw_Guid00018_Pdu01800_f7l
Tp Threshold:

value == Tx_IfGw_Guid00018_Pdu01800_f7bea999
3 items found

/ActiveEcuC/CanIf/CanIfInitCfg/Tx_IfGw_Guid00018_Pdu01800_oCAN001_Tx_3abd61be[0:CanIfTxPduRef]
/ActiveEcuC/PduR/PduRRoutingTables/PduRRoutingTable/PduRRoutingPath_PduRSrcPdu_Rx_IfGw_Guid00017_Pdu01800_adcbdbd3_To_PduRDestPdu_Tx_IfGw_Guid00018_Pdu01800_f7bea999/PduRDestPdu_Tx_IfGw_Guid00018_Pdu01800_f7bea999
/ActiveEcuC/PduR/PduRRoutingTables/PduRRoutingTable/PduRRoutingPath_PduRSrcPdu_Rx_IfGw_Guid00017_Pdu018 … w_Guid00018_Pdu01800_f7bea999/PduRDestPdu_Tx_IfGw_Guid00018_Pdu01800_f7bea999

Table 3-6 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.

3.6.8 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.

3.7 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.

3.7.1 Multi-Routing

The PduR supports N:1 and 1:N routing paths for both single- and multiframes. 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 bufferSizes 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.

3.7.2 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 3.8.3.1 for details.

3.7.2.1 Restrictions

The setting of the TP Threshold is restricted for CanTp, LinTp and FrArTp routing paths. Threshold values in the shaded sections of Figure 3-9 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 3-9 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.

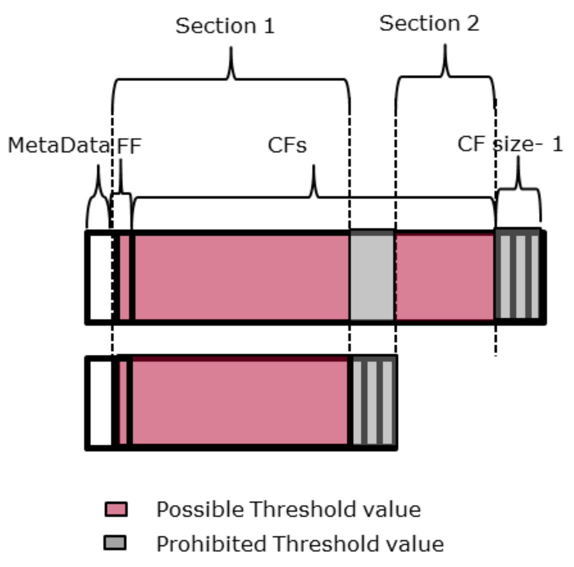


Figure 3-9 Configured Tp Buffer with possible Threshold ranges

3.7.2.2 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 [6].

3.7.3 Queued TP routing

3.7.3.1 Supported Queues

A transport protocol routing path only supports the Shared Buffer Queue. (see page 30)

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.

3.7.3.2 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.

3.7.4 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.

3.7.5 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.

3.8 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

3.8.1 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.

3.8.2 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.

3.8.3 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.

3.8.3.1 Tx Buffer Assignment

3.8.3.1.1 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.

3.8.3.1.2 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.

3.8.3.2 Tx Buffer Length Configuration

3.8.3.2.1 Communication Interface

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

3.8.3.2.2 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.

3.8.3.3 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 availabe 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.

3.9 Cross Partition Routing

3.9.1 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.

Each PduRBswModule can be assigned to a partition. This is done by referring the corresponding OsApplication in `/MICROSAR/PduR/PduRBswModules /PduROsApplicationRef`. This parameter is optional and shall be left empty for single-partition configurations.

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

The memory section configuration is described in chapter 4.3.2.

3.9.2 Multicore Queue

A specialized multicore queue (McQ) is used to handle the efficient routing between different 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.

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.

3.9.3 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`.

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.

3.9.4 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 3.9.2).

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)

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 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 OsApplications of different ASIL.

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.

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

3.9.5 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 and queue depth of one) 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.

- 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.

4 Integration

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

4.1 Scope of Delivery

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

4.1.1 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_Sm.c	■	This is the source file of the PDUR Switching Manager
PduR_Sm.h	■	This is the header file of the PDUR Switching 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_RmIf.c	■	This is the source file of the PDUR Routing Manager If
PduR_RmIf.h	■	This is the header file of the PDUR Routing Manager If
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
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_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

Table 4-1 Static files

4.1.2 Dynamic Files

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

File Name	Description
PduR_Cfg.h	This file contains:

File Name	Description
	▶ global constant macros ▶ global function macros ▶ global data types and structures ▶ global data prototypes ▶ global function prototypes of CONFIG-CLASS PRE-COMPILE data.
PduR_Lcfg.h	This file contains: ▶ global constant macros ▶ global function macros ▶ global data types and structures ▶ global data prototypes ▶ global function prototypes of CONFIG-CLASS LINK data.
PduR_Lcfg.c	This file contains: ▶ local constant macros ▶ local function macros ▶ local data types and structures ▶ local data prototypes ▶ local data ▶ global data of CONFIG-CLASS LINK and PRE-COMPILE data.
PduR_PBcfg.h	This file contains: ▶ global constant macros ▶ global function macros ▶ global data types and structures ▶ global data prototypes ▶ global function prototypes of CONFIG-CLASS POST-BUILD data.
PduR_PBcfg.c	This file contains: ▶ local constant macros ▶ local function macros ▶ local data types and structures ▶ local data prototypes ▶ local data ▶ global data of CONFIG-CLASS POST-BUILD data.
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 4-2 Generated files

4.2 Critical Sections

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

4.3 Memory Sections

4.3.1 Memory sections for PduRTxBuffer

With PDUR_START_SEC_BUFFER_VAR_NOINIT_8BIT and PDUR_STOP_SEC_BUFFER_VAR_NOINIT_8BIT large Tx buffer can be mapped into an own memory section.

4.3.2 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 4-3.

Memory Section	Partition_X	Partition_Y
PDUR_SEC_PARTITION_X_<SectionType>	RW	RO
PDUR_SEC_PARTITION_Y_<SectionType>	RO	RW
PDUR_SEC_COMMONSHAREDMEMORY_<SectionType>	RW	RW

Table 4-3 Memory Mapping in multi-partition use case

4.4 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 4-4 Type definitions

5 API Description

5.1 Services provided by PDUR

5.1.1 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 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	
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. In Multi Core UseCse: This function shall only be called ONCE and NOT for each core/OsApplication.	

5.1.2 PduR_Init

Prototype	
`void PduR_Init (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 initializes the PDU Router in the context of the calling OsApplication. If the initialization is performed successfully the PDU Router on this 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 5-1 PduR_Init

5.1.3 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 5-2 PduR_InitMemory

5.2 Services

5.2.1 PduR_GetVersionInfo

Prototype	
void **PduR_GetVersionInfo** (Std_VersionInfoType* versioninfo)	
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 5-3 PduR_GetVersionInfo

5.2.2 PduR_GetConfigurationId

Prototype	
uint32 **PduR_GetConfigurationId** (void)	
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 5-4 PduR_GetConfigurationId

5.2.3 PduR_EnableRouting

Prototype	
`void PduR_EnableRouting (PduR_RoutingPathGroupIdType id)`	
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 5-5 PduR_EnableRouting

5.2.4 PduR_DisableRouting

Prototype	
`void PduR_DisableRouting (PduR_RoutingPathGroupIdType id)`	
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 5-6 PduR_DisableRouting

5.3 Communication Interface

5.3.1 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 5-7 PduR_<GenericUp>Transmit

5.3.2 PduR_<GenericLo>RxIndication

Prototype	
`void` **`PduR_<GenericLo>RxIndication`** `(PduIdType id, const PduInfoType* info)`	
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	
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.	
Particularities and Limitations	
The function is called by a lower communication interface module.	
Call Context	
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.	

Table 5-8 PduR_<GenericLo>RxIndication

5.3.3 PduR_<GenericLo>TriggerTransmit

Prototype	
`void PduR_<GenericLo>TriggerTransmit (PduIdType id, PduInfoType* info)`	
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 SduLengh. 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 TxPduId 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 5-9 PduR_<GenericLo>TriggerTransmit

5.3.4 PduR_<GenericLo>TxConfirmation

Prototype	
void **PduR_<GenericLo>TxConfirmation** (PduIdType id)	
Parameter	
id	Handle ID of the transmitted lower layer communication interface I-PDU.
Return code	
void	none
Functional Description	
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.	
Particularities and Limitations	
The function is called by a lower communication interface module.	
Call Context	
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.	

Table 5-10 PduR_<GenericLo>TxConfirmation

5.4 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.

5.4.1 PduR_<GenericUpTp>ChangeParameter

Prototype	
`Std_ReturnType` **`PduR_<GenericUpTp>ChangeParameter`** `(PduIdType id, TPParameterType parameter, uint16 value)`	
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	Std_ReturnType E_OK: The parameter was changed successfully. E_NOT_OK: The parameter change was rejected.
Functional Description	
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.	
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 5-11 PduR_<GenericUpTp>ChangeParameter

5.4.2 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 5-12 PduR_<GenericUpTp>CancelReceive

5.4.3 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 5-13 PduR_<GenericUpTp>CancelTransmit

5.4.4 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 5-14 PduR_<GenericLoTp>StartOfReception

5.4.5 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 5-15 PduR_<GenericLoTp>CopyRxData

5.4.6 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 5-16 PduR_<GenericLoTp>CopyTxData

5.4.7 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	Result of the TP transmission E_OK The TP transmission has been completed successfully. 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.
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 5-17 PduR_<GenericLo>TpTxConfirmation

5.4.8 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	Result of the TP reception E_OK The TP reception has been completed successfully. E_NOT_OK PduR_Init() has not been called or the reception 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.
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 5-18 PduR_<GenericLo>TpRxIndication

5.4.9 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	
The function serves to request the transmission of a TP layer I-PDU. 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.	
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 5-19 PduR_<GenericUpTp>Transmit

5.5 Service Ports

5.5.1 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.

6 Configuration

6.1 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 6-1 visualizes the range-routing technique.

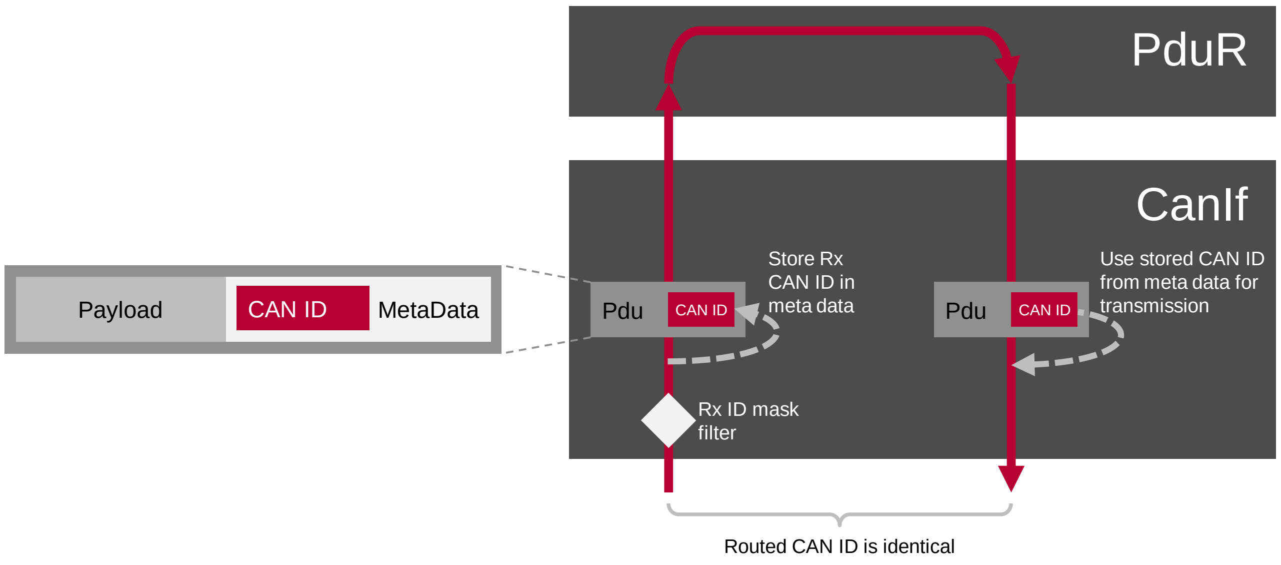

PduR
CanIf
Payload
CAN ID
MetaData
Pdu
CAN ID
Store Rx CAN ID in meta data
Pdu
CAN ID
Use stored CAN ID from meta data for transmission
Rx ID mask filter
Routed CAN ID is identical

Figure 6-1 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

6.1.1 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 6-1 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 6-2.

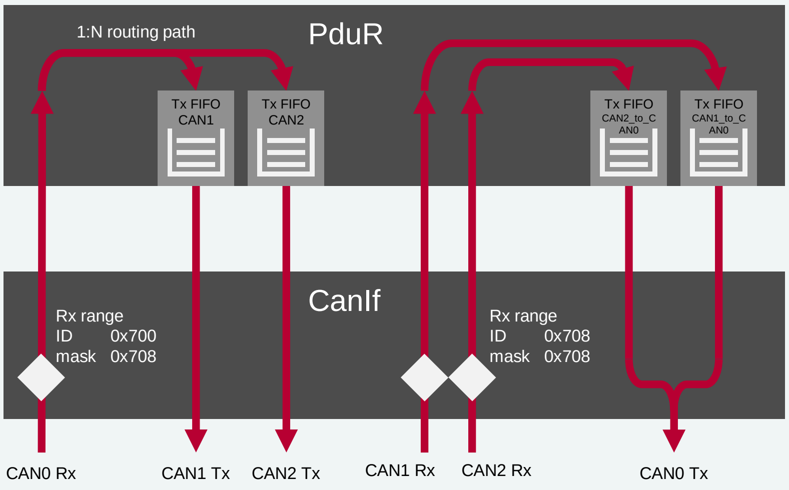
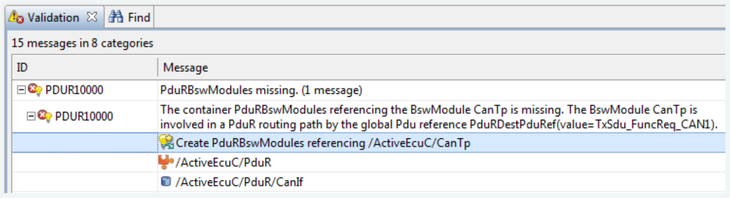


Figure 6-2 CanIf / PduR range routing example overview

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.

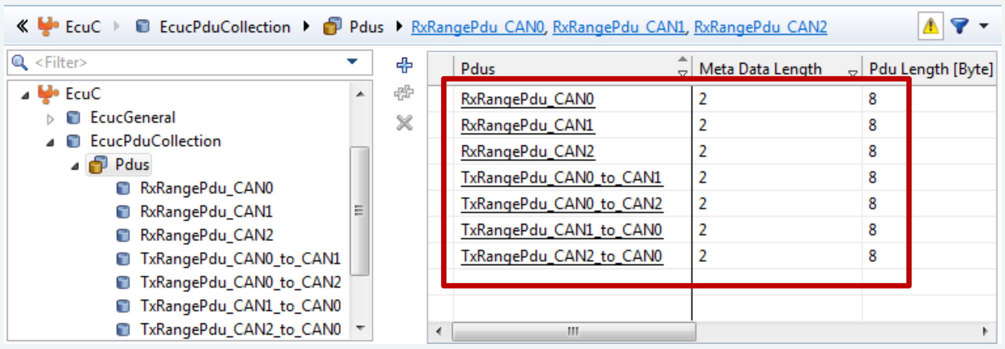

Validation Find

15 messages in 8 categories

ID	Message
PDUR10000	PduRBswModules missing. (1 message)
PDUR10000	The container PduRBswModules referencing the BswModule CanTp is missing. The BswModule CanTp is involved in a PduR routing path by the global Pdu reference PduRDestPduRef(value=TxSdu_FuncReq_CAN1).
	Create PduRBswModules referencing /ActiveEcuC/CanTp
	/ActiveEcuC/PduR
	/ActiveEcuC/PduR/CanIf

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:

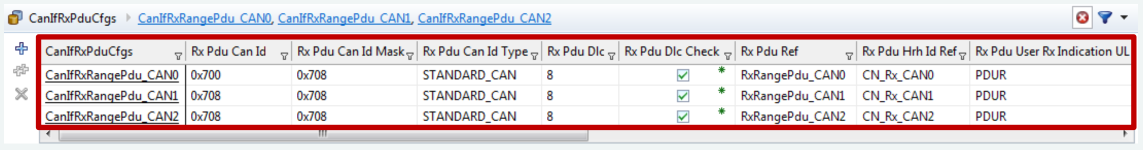


Pdus	Meta Data Length	Pdu Length [Byte]
RxRangePdu_CAN0	2	8
RxRangePdu_CAN1	2	8
RxRangePdu_CAN2	2	8
TxRangePdu_CAN0_to_CAN1	2	8
TxRangePdu_CAN0_to_CAN2	2	8
TxRangePdu_CAN1_to_CAN0	2	8
TxRangePdu_CAN2_to_CAN0	2	8

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:

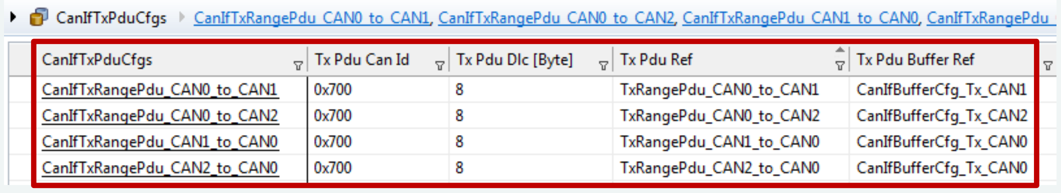

CanIfRxPduCfgs ▸ CanIfRxRangePdu_CAN0, CanIfRxRangePdu_CAN1, CanIfRxRangePdu_CAN2

CanIfRxPduCfgs	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	✓	RxRangePdu_CAN0	CN_Rx_CAN0	PDUR
CanIfRxRangePdu_CAN1	0x708	0x708	STANDARD_CAN	8	✓	RxRangePdu_CAN1	CN_Rx_CAN1	PDUR
CanIfRxRangePdu_CAN2	0x708	0x708	STANDARD_CAN	8	✓	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_

CanIfTxPduCfgs	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

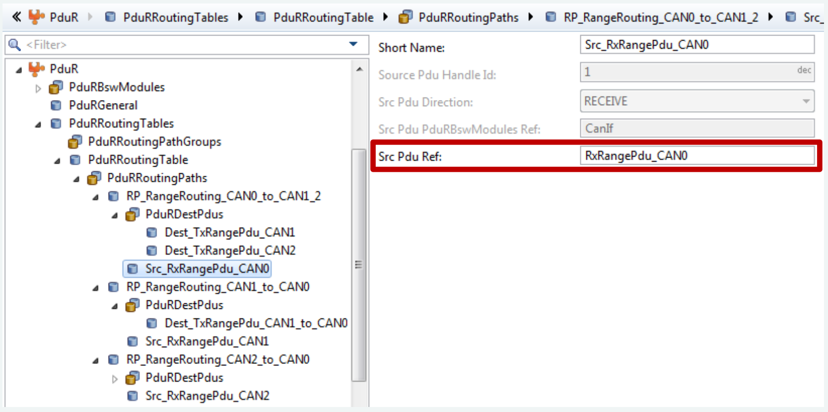
Create PduR routing paths

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`

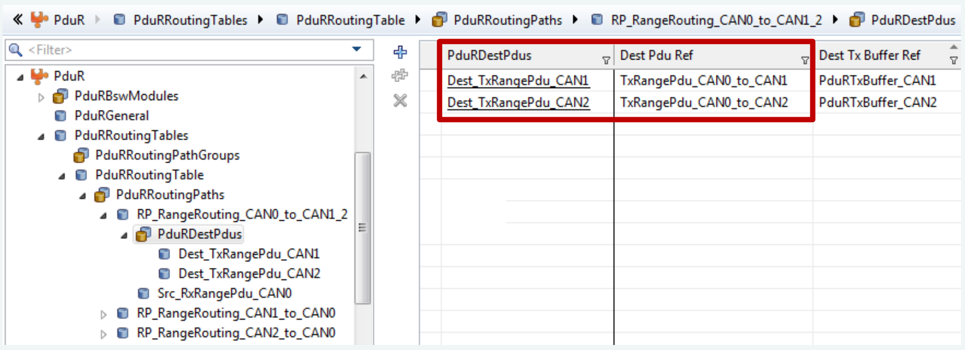
For the example configuration this results in the following PduR routing path configuration:

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


PduR ▸ PduRRoutingTables ▸ PduRRoutingTable ▸ PduRRoutingPaths ▸ RP_RangeRouting_CAN0_to_CAN1_2 ▸ Src_

```
PduR
  PduRBswModules
  PduRGeneral
  PduRRoutingTables
    PduRRoutingPathGroups
    PduRRoutingTable
      PduRRoutingPaths
        RP_RangeRouting_CAN0_to_CAN1_2
          PduRDestPdus
            Dest_TxRangePdu_CAN1
            Dest_TxRangePdu_CAN2
          Src_RxRangePdu_CAN0
        RP_RangeRouting_CAN1_to_CAN0
          PduRDestPdus
            Dest_TxRangePdu_CAN1_to_CAN0
          Src_RxRangePdu_CAN1
        RP_RangeRouting_CAN2_to_CAN0
          PduRDestPdus
          Src_RxRangePdu_CAN2
```

Short Name:	Src_RxRangePdu_CAN0
Source Pdu Handle Id:	1 dec
Src Pdu Direction:	RECEIVE
Src Pdu PduRBswModules Ref:	CanIf
Src Pdu Ref:	RxRangePdu_CAN0


PduR ▸ PduRRoutingTables ▸ PduRRoutingTable ▸ PduRRoutingPaths ▸ RP_RangeRouting_CAN0_to_CAN1_2 ▸ PduRDestPdus

```
PduR
  PduRBswModules
  PduRGeneral
  PduRRoutingTables
    PduRRoutingPathGroups
    PduRRoutingTable
      PduRRoutingPaths
        RP_RangeRouting_CAN0_to_CAN1_2
          PduRDestPdus
            Dest_TxRangePdu_CAN1
            Dest_TxRangePdu_CAN2
          Src_RxRangePdu_CAN0
        RP_RangeRouting_CAN1_to_CAN0
        RP_RangeRouting_CAN2_to_CAN0
```

PduRDestPdus	Dest Pdu Ref	Dest Tx Buffer Ref
Dest_TxRangePdu_CAN1	TxRangePdu_CAN0_to_CAN1	PduRTxBuffer_CAN1
Dest_TxRangePdu_CAN2	TxRangePdu_CAN0_to_CAN2	PduRTxBuffer_CAN2

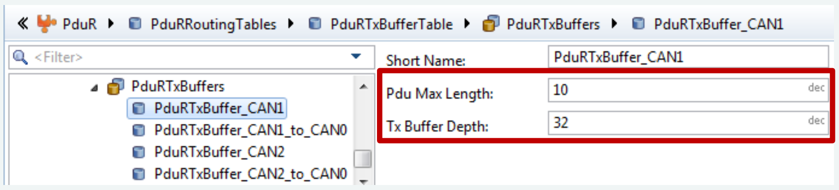
For further details regarding the PduR communication interface gateway, please refer to chapter 3.6.

6.1.2 Optional configuration variants / options

[Optional] Configure PduR FIFO routing

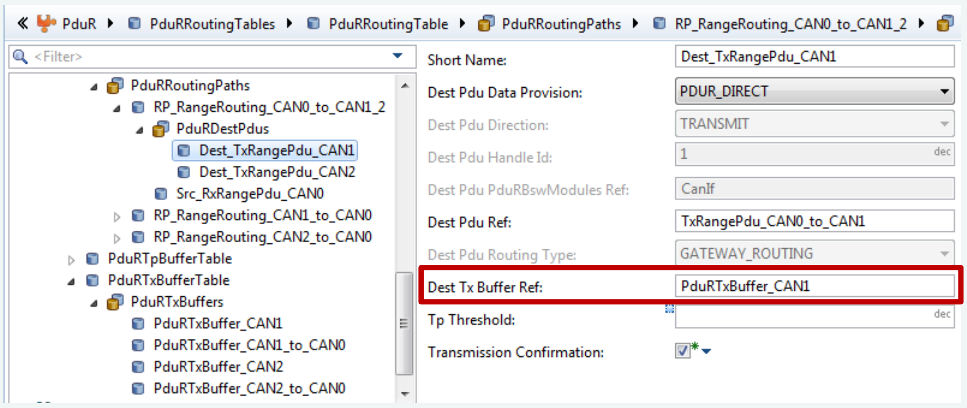
In case the sequence of successive routed range Pdus shall be retained, a FIFO queue must be configured within the PduR.

- 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 3.6.2): `/MICROSAR/PduR/PduRRoutingTables/PduRTxBufferTable/PduRTxBuffer/PduRTxBufferDepth`


PduR ▸ PduRRoutingTables ▸ PduRTxBufferTable ▸ PduRTxBuffers ▸ PduRTxBuffer_CAN1
<Filter>
PduRTxBuffers
PduRTxBuffer_CAN1
PduRTxBuffer_CAN1_to_CAN0
PduRTxBuffer_CAN2
PduRTxBuffer_CAN2_to_CAN0

Short Name:	PduRTxBuffer_CAN1
Pdu Max Length:	10
Tx Buffer Depth:	32

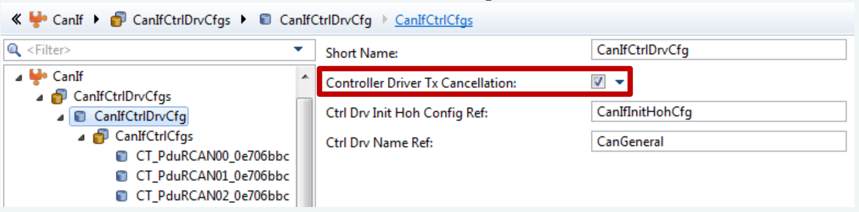
- Reference the created Tx buffers at the related PduR range routing paths: `/MICROSAR/PduR/PduRRoutingTables/PduRRoutingTable/PduRRoutingPath/PduRDestPdu/PduRDestTxBufferRef`


PduR ▸ PduRRoutingTables ▸ PduRRoutingTable ▸ PduRRoutingPaths ▸ RP_RangeRouting_CAN0_to_CAN1_2 ▸
<Filter>
PduRRoutingPaths
RP_RangeRouting_CAN0_to_CAN1_2
PduRDestPdus
Dest_TxRangePdu_CAN1
Dest_TxRangePdu_CAN2
Src_RxRangePdu_CAN0
RP_RangeRouting_CAN1_to_CAN0
RP_RangeRouting_CAN2_to_CAN0
PduRTpBufferTable
PduRTxBufferTable
PduRTxBuffers
PduRTxBuffer_CAN1
PduRTxBuffer_CAN1_to_CAN0
PduRTxBuffer_CAN2
PduRTxBuffer_CAN2_to_CAN0

Short Name:	Dest_TxRangePdu_CAN1
Dest Pdu Data Provision:	PDUR_DIRECT
Dest Pdu Direction:	TRANSMIT
Dest Pdu Handle Id:	1
Dest Pdu PduRBswModules Ref:	CanIf
Dest Pdu Ref:	TxRangePdu_CAN0_to_CAN1
Dest Pdu Routing Type:	GATEWAY_ROUTING
Dest Tx Buffer Ref:	PduRTxBuffer_CAN1
Tp Threshold:	
Transmission Confirmation:	✓

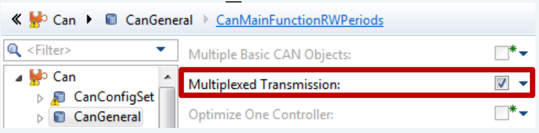
[Optional] CAN priority inversion

- Enable the CanIf feature „Cancellation of PDUs and requeueing” in order to avoid inner priority inversion:
  `/MICROSAR/CanIf/CanIfCtrlDrvCfg/CanIfCtrlDrvTxCancellation`



Short Name: CanIfCtrlDrvCfg
Controller Driver Tx Cancellation: ☑
Ctrl Drv Init Hoh Config Ref: CanIfInitHohCfg
Ctrl Drv Name Ref: CanGeneral

- Enable the CAN driver feature „Multiplexed Transmission“ in order to avoid external priority inversion:
  `/MICROSAR/<CAN_Platform>/Can/CanGeneral/CanMultiplexedTransmission`



Multiple Basic CAN Objects: ☐
Multiplexed Transmission: ☑
Optimize One Controller: ☐

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

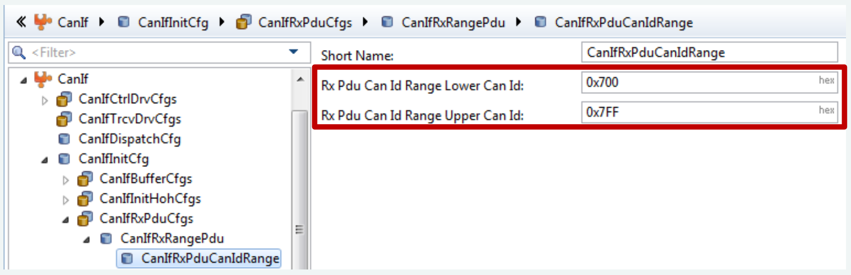
Hint: These features are not supported by all CAN controllers. Please refer to [7] and [8].

[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`



Short Name: CanIfRxPduCanIdRange
Rx Pdu Can Id Range Lower Can Id: 0x700 hex
Rx Pdu Can Id Range Upper Can Id: 0x7FF hex

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 [7] and [8].

6.2 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.

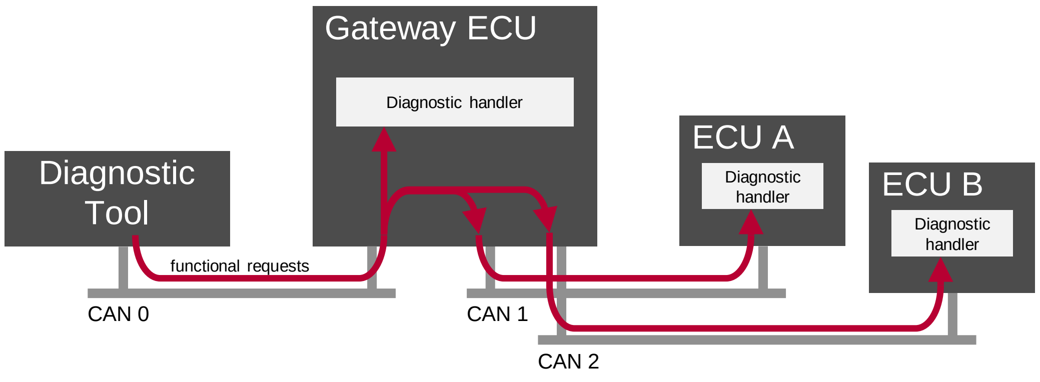


Figure 6-3 Example functional requests gateway network architecture

To realize the gateway behavior as visualized in Figure 6-3, a PduR 1:N transport protocol gateway could be configured, as shown in Figure 6-4.

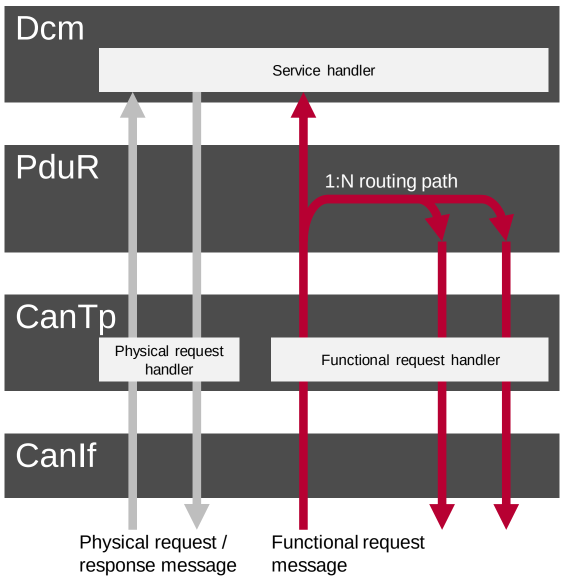

Dcm
Service handler
PduR
1:N routing path
CanTp
Physical request handler
Functional request handler
CanIf
Physical request / response message
Functional request message

Figure 6-4 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 6.1 introducing range-routing paths which could be used for efficient and simple routing of physically addressed diagnostic messages.

6.2.1 Step-by-step configuration

Configuration Example

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

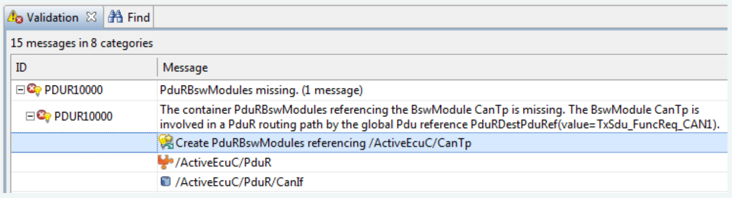
Functional diagnostic request ID	DLC	Source network / channel	Destination network(s) / channel(s)
0x7DF	8	CAN0	CAN1, CAN2

Table 6-2 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.



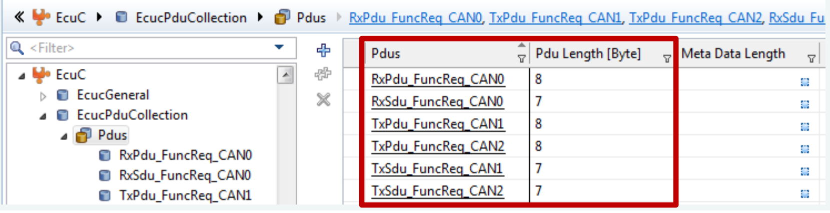
15 messages in 8 categories

ID	Message
PDUR10000	PduRBswModules missing. (1 message)
PDUR10000	The container PduRBswModules referencing the BswModule CanTp is missing. The BswModule CanTp is involved in a PduR routing path by the global Pdu reference PduRDestPduRef(value=TxSdu_FuncReq_CAN1).
	Create PduRBswModules referencing /ActiveEcuC/CanTp
	/ActiveEcuC/PduR
	/ActiveEcuC/PduR/CanIf

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:



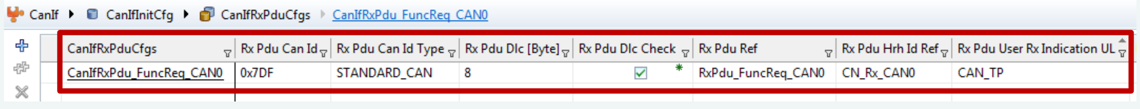
Pdus	Pdu Length [Byte]	Meta Data Length
RxPdu_FuncReq_CAN0	8	
RxSdu_FuncReq_CAN0	7	
TxPdu_FuncReq_CAN1	8	
TxPdu_FuncReq_CAN2	8	
TxSdu_FuncReq_CAN1	7	
TxSdu_FuncReq_CAN2	7	

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:

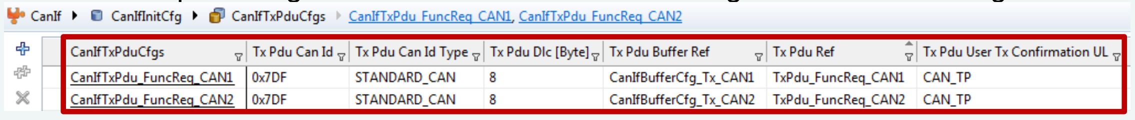

CanIf ▸ CanIfInitCfg ▸ CanIfRxPduCfgs ▸ CanIfRxPdu_FuncReq_CAN0

CanIfRxPduCfgs	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	Rx Pdu User Rx Indication UL
CanIfRxPdu_FuncReq_CAN0	0x7DF	STANDARD_CAN	8	✓	RxPdu_FuncReq_CAN0	CN_Rx_CAN0	CAN_TP

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:

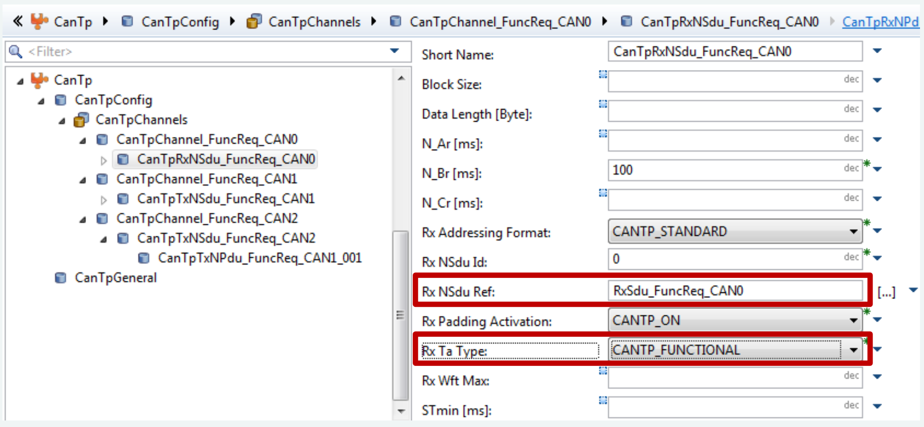

CanIf ▸ CanIfInitCfg ▸ CanIfTxPduCfgs ▸ CanIfTxPdu_FuncReq_CAN1, CanIfTxPdu_FuncReq_CAN2

CanIfTxPduCfgs	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`

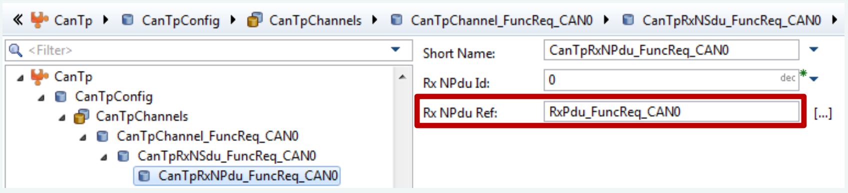

CanTp ▸ CanTpConfig ▸ CanTpChannels ▸ CanTpChannel_FuncReq_CAN0 ▸ CanTpRxNSdu_FuncReq_CAN0 ▸ CanTpRxNPd

<Filter>

```
CanTp
  CanTpConfig
    CanTpChannels
      CanTpChannel_FuncReq_CAN0
        CanTpRxNSdu_FuncReq_CAN0
      CanTpChannel_FuncReq_CAN1
        CanTpTxNSdu_FuncReq_CAN1
      CanTpChannel_FuncReq_CAN2
        CanTpTxNSdu_FuncReq_CAN2
          CanTpTxNPdu_FuncReq_CAN1_001
  CanTpGeneral
```

Short Name:	CanTpRxNSdu_FuncReq_CAN0
Block Size:	
Data Length [Byte]:	
N_Ar [ms]:	
N_Br [ms]:	100
N_Cr [ms]:	
Rx Addressing Format:	CANTP_STANDARD
Rx NSdu Id:	0
Rx NSdu Ref:	RxSdu_FuncReq_CAN0
Rx Padding Activation:	CANTP_ON
Rx Ta Type:	CANTP_FUNCTIONAL
Rx Wft Max:	
STmin [ms]:	

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


CanTp ▸ CanTpConfig ▸ CanTpChannels ▸ CanTpChannel_FuncReq_CAN0 ▸ CanTpRxNSdu_FuncReq_CAN0 ▸

<Filter>

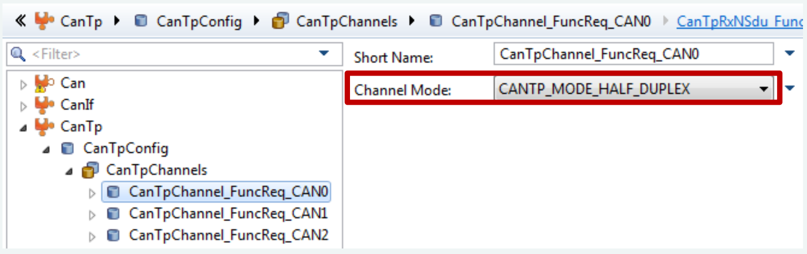
```
CanTp
  CanTpConfig
    CanTpChannels
      CanTpChannel_FuncReq_CAN0
        CanTpRxNSdu_FuncReq_CAN0
          CanTpRxNPdu_FuncReq_CAN0
```

Short Name:	CanTpRxNPdu_FuncReq_CAN0
Rx NPdu Id:	0
Rx NPdu Ref:	RxPdu_FuncReq_CAN0

For further information regarding the CanTp timing configuration parameters please refer to [9].

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`



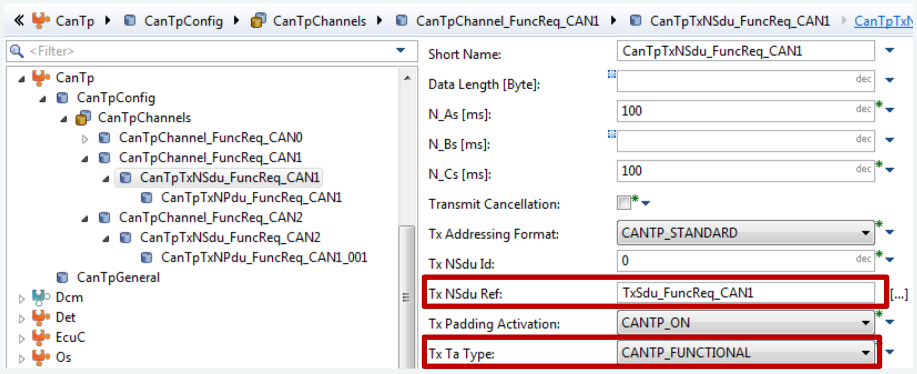
« CanTp ▸ CanTpConfig ▸ CanTpChannels ▸ CanTpChannel_FuncReq_CAN0 ▸ CanTpRxNSdu_Func

<Filter>

- Can
- CanIf
- CanTp
  - CanTpConfig
    - CanTpChannels
      - CanTpChannel_FuncReq_CAN0
      - CanTpChannel_FuncReq_CAN1
      - CanTpChannel_FuncReq_CAN2

Short Name:	CanTpChannel_FuncReq_CAN0
Channel Mode:	CANTP_MODE_HALF_DUPLEX

- 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`



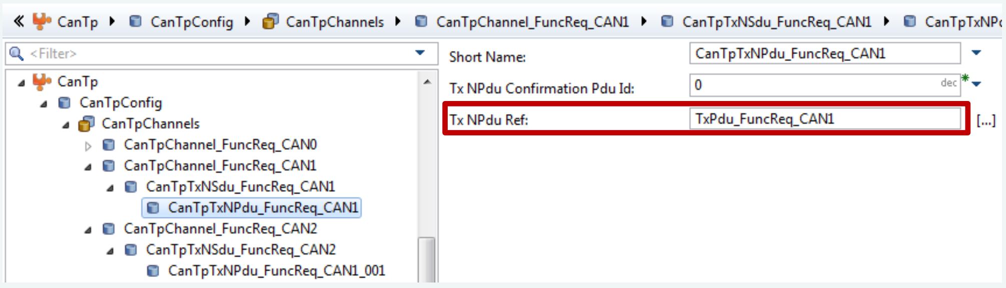
« CanTp ▸ CanTpConfig ▸ CanTpChannels ▸ CanTpChannel_FuncReq_CAN1 ▸ CanTpTxNSdu_FuncReq_CAN1 ▸ CanTpTxN

<Filter>

- CanTp
  - CanTpConfig
    - CanTpChannels
      - CanTpChannel_FuncReq_CAN0
      - CanTpChannel_FuncReq_CAN1
        - CanTpTxNSdu_FuncReq_CAN1
          - CanTpTxNPdu_FuncReq_CAN1
      - CanTpChannel_FuncReq_CAN2
        - CanTpTxNSdu_FuncReq_CAN2
          - CanTpTxNPdu_FuncReq_CAN1_001
  - CanTpGeneral
- Dcm
- Det
- EcuC
- Os

Short Name:	CanTpTxNSdu_FuncReq_CAN1
Data Length [Byte]:	
N_As [ms]:	100
N_Bs [ms]:	
N_Cs [ms]:	100
Transmit Cancellation:	☐
Tx Addressing Format:	CANTP_STANDARD
Tx NSdu Id:	0
Tx NSdu Ref:	TxSdu_FuncReq_CAN1
Tx Padding Activation:	CANTP_ON
Tx Ta Type:	CANTP_FUNCTIONAL

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



« CanTp ▸ CanTpConfig ▸ CanTpChannels ▸ CanTpChannel_FuncReq_CAN1 ▸ CanTpTxNSdu_FuncReq_CAN1 ▸ CanTpTxNP

<Filter>

- CanTp
  - CanTpConfig
    - CanTpChannels
      - CanTpChannel_FuncReq_CAN0
      - CanTpChannel_FuncReq_CAN1
        - CanTpTxNSdu_FuncReq_CAN1
          - CanTpTxNPdu_FuncReq_CAN1
      - CanTpChannel_FuncReq_CAN2
        - CanTpTxNSdu_FuncReq_CAN2
          - CanTpTxNPdu_FuncReq_CAN1_001

Short Name:	CanTpTxNPdu_FuncReq_CAN1
Tx NPdu Confirmation Pdu Id:	0
Tx NPdu Ref:	TxPdu_FuncReq_CAN1

For further information regarding the CanTp timing configuration parameters please refer to [9].

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 3.7.3 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`

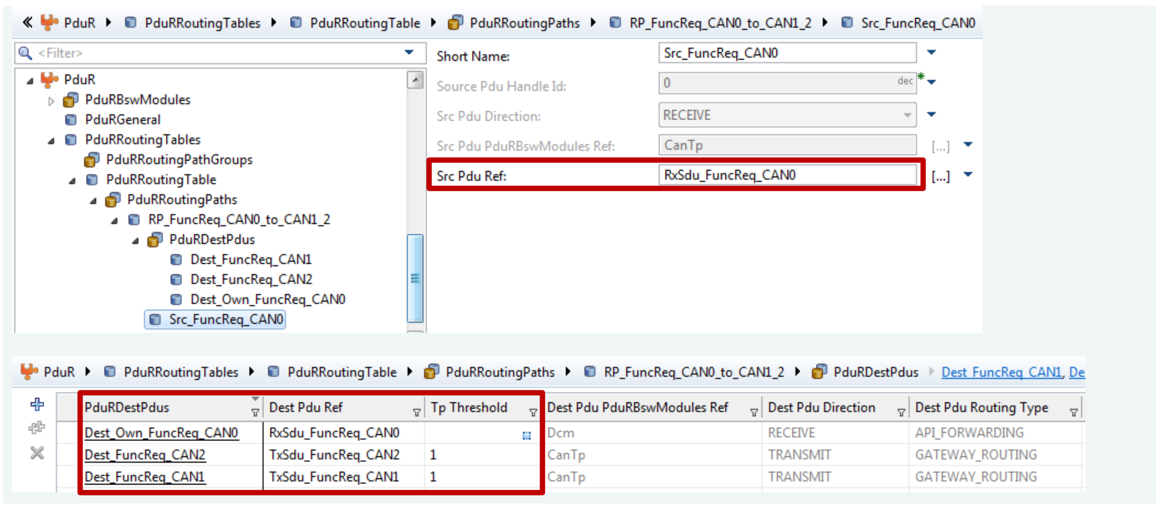
  and the routing path destination:
  `/MICROSAR/PduR/PduRRoutingTables/PduRRoutingTable/PduRRoutingPath/PduRDestPdu/PduRDestPduRef`

  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 3.7.2 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



PduR ▸ PduRRoutingTables ▸ PduRRoutingTable ▸ PduRRoutingPaths ▸ RP_FuncReq_CAN0_to_CAN1_2 ▸ Src_FuncReq_CAN0

- PduR
  - PduRBswModules
  - PduRGeneral
  - PduRRoutingTables
    - PduRRoutingPathGroups
    - PduRRoutingTable
      - PduRRoutingPaths
        - RP_FuncReq_CAN0_to_CAN1_2
          - PduRDestPdus
            - Dest_FuncReq_CAN1
            - Dest_FuncReq_CAN2
            - Dest_Own_FuncReq_CAN0
          - Src_FuncReq_CAN0

Short Name:	Src_FuncReq_CAN0
Source Pdu Handle Id:	0
Src Pdu Direction:	RECEIVE
Src Pdu PduRBswModules Ref:	CanTp
Src Pdu Ref:	RxSdu_FuncReq_CAN0

PduR ▸ PduRRoutingTables ▸ PduRRoutingTable ▸ PduRRoutingPaths ▸ RP_FuncReq_CAN0_to_CAN1_2 ▸ PduRDestPdus ▸ Dest_FuncReq_CAN1, De

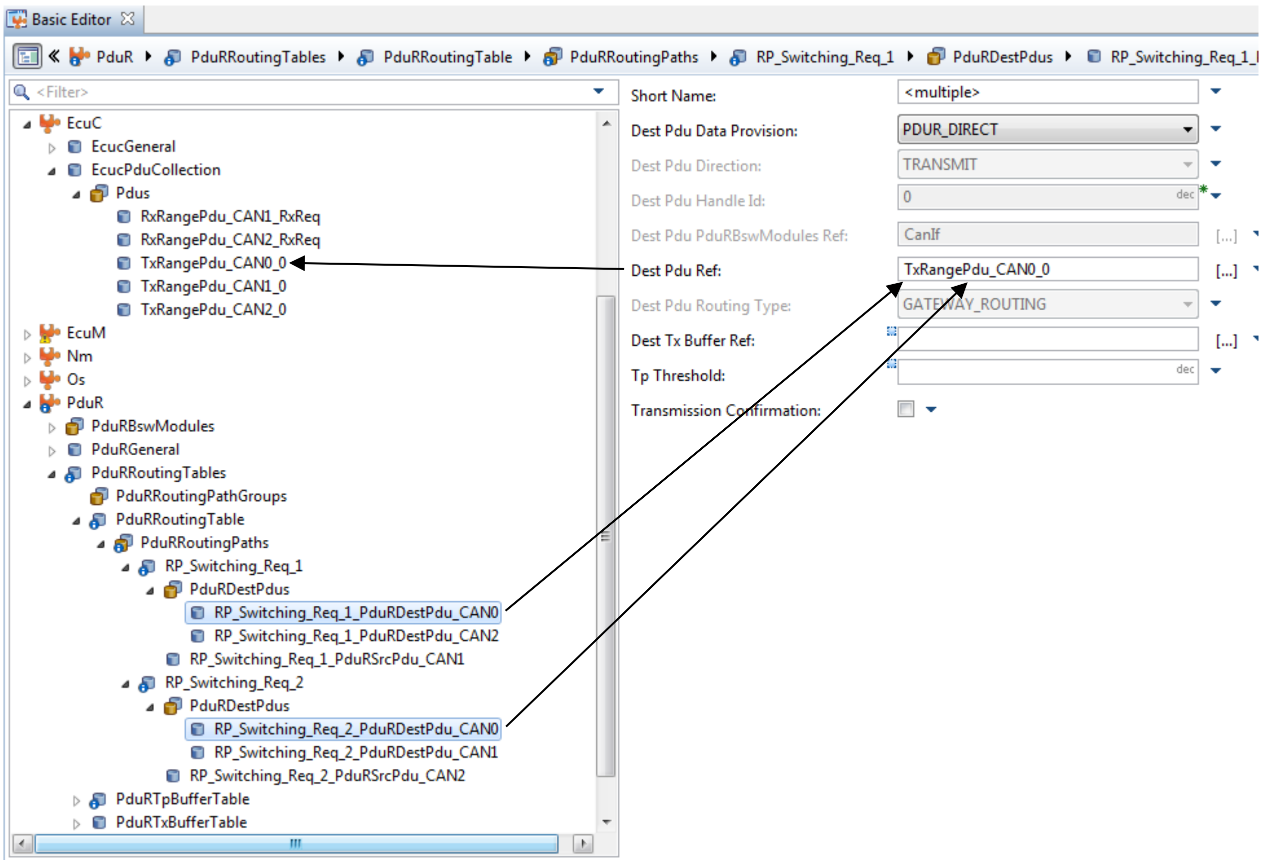
PduRDestPdus	Dest Pdu Ref	Tp Threshold	Dest Pdu PduRBswModules Ref	Dest Pdu Direction	Dest Pdu Routing Type
Dest_Own_FuncReq_CAN0	RxSdu_FuncReq_CAN0		Dcm	RECEIVE	API_FORWARDING
Dest_FuncReq_CAN2	TxSdu_FuncReq_CAN2	1	CanTp	TRANSMIT	GATEWAY_ROUTING
Dest_FuncReq_CAN1	TxSdu_FuncReq_CAN1	1	CanTp	TRANSMIT	GATEWAY_ROUTING

For further details about the CanIf and CanTp modules, please refer to [7] and [9].

6.3 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 6-5. 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 6-6. The names refer to the example configuration in Figure 6-5.

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


Basic Editor
PduR ▸ PduRRoutingTables ▸ PduRRoutingTable ▸ PduRRoutingPaths ▸ RP_Switching_Req_1 ▸ PduRDestPdus ▸ RP_Switching_Req_1_
<Filter>

- EcuC
  - EcucGeneral
  - EcucPduCollection
    - Pdus
      - RxRangePdu_CAN1_RxReq
      - RxRangePdu_CAN2_RxReq
      - TxRangePdu_CAN0_0
      - TxRangePdu_CAN1_0
      - TxRangePdu_CAN2_0
- EcuM
- Nm
- Os
- PduR
  - PduRBswModules
  - PduRGeneral
  - PduRRoutingTables
    - PduRRoutingPathGroups
    - PduRRoutingTable
      - PduRRoutingPaths
        - RP_Switching_Req_1
          - PduRDestPdus
            - RP_Switching_Req_1_PduRDestPdu_CAN0
            - RP_Switching_Req_1_PduRDestPdu_CAN2
          - RP_Switching_Req_1_PduRSrcPdu_CAN1
        - RP_Switching_Req_2
          - PduRDestPdus
            - RP_Switching_Req_2_PduRDestPdu_CAN0
            - RP_Switching_Req_2_PduRDestPdu_CAN1
          - RP_Switching_Req_2_PduRSrcPdu_CAN2
    - PduRTpBufferTable
    - PduRTxBufferTable

Short Name:	<multiple>
Dest Pdu Data Provision:	PDUR_DIRECT
Dest Pdu Direction:	TRANSMIT
Dest Pdu Handle Id:	0
Dest Pdu PduRBswModules Ref:	CanIf
Dest Pdu Ref:	TxRangePdu_CAN0_0
Dest Pdu Routing Type:	GATEWAY_ROUTING
Dest Tx Buffer Ref:	
Tp Threshold:	
Transmission Confirmation:	☐

Figure 6-5 example N:1 routing path configuration

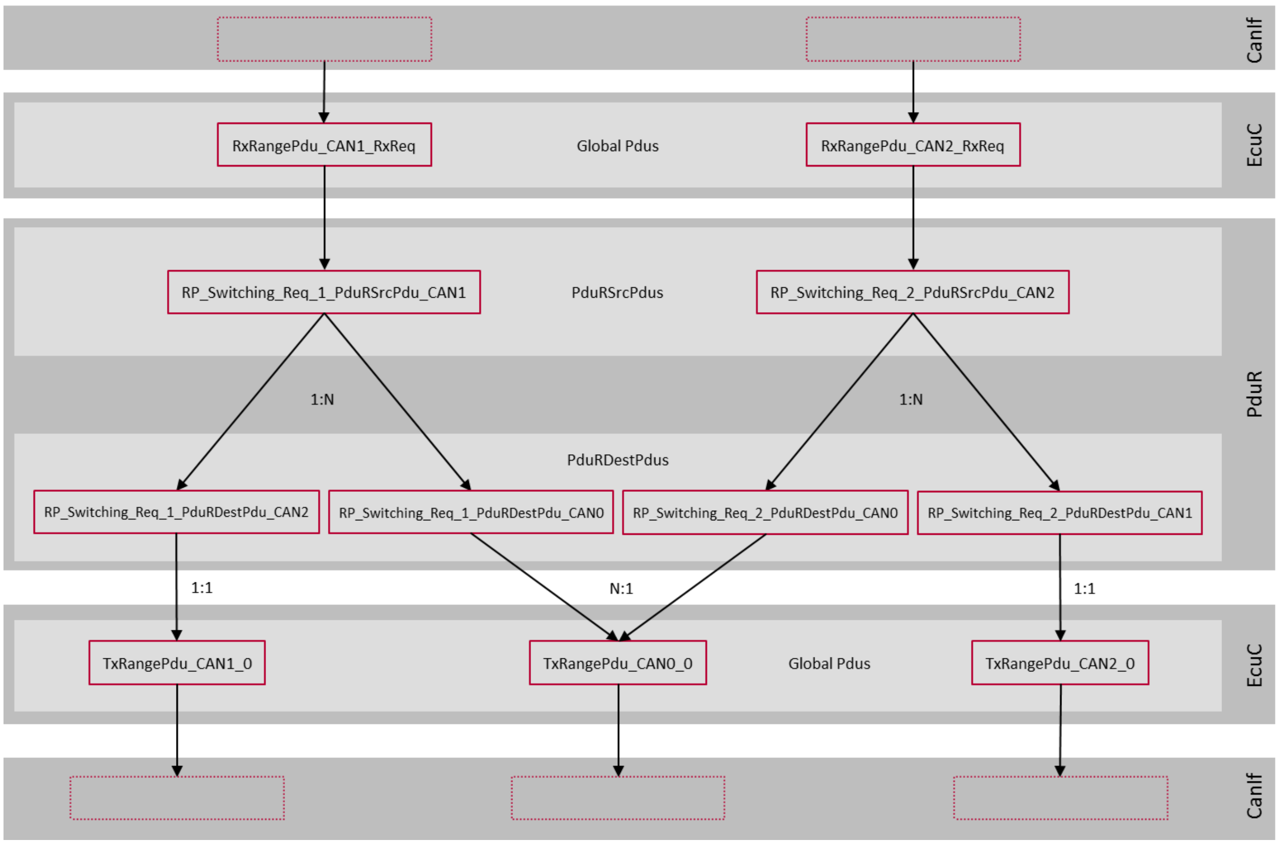

CanIf
RxRangePdu_CAN1_RxReq
Global Pdus
RxRangePdu_CAN2_RxReq
EcuC
RP_Switching_Req_1_PduRSrcPdu_CAN1
PduRSrcPdus
RP_Switching_Req_2_PduRSrcPdu_CAN2
1:N
1:N
PduR
PduRDestPdus
RP_Switching_Req_1_PduRDestPdu_CAN2
RP_Switching_Req_1_PduRDestPdu_CAN0
RP_Switching_Req_2_PduRDestPdu_CAN0
RP_Switching_Req_2_PduRDestPdu_CAN1
1:1
N:1
1:1
TxRangePdu_CAN1_0
TxRangePdu_CAN0_0
Global Pdus
TxRangePdu_CAN2_0
EcuC
CanIf

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

7 AUTOSAR Standard Compliance

7.1 Deviations

- The API PduR_<User:LoTp>StartOfReception is implemented according to SWS_PduR_00507 ASR 4.1.2 [3]
  - The PduR does not provide the return value "BUFREQ_E_BUSY" for the function PduR_<User:LoTp>StartOfReception like specified in Requirement PDUR507 in [1].
  - The parameter list contains a PduInfoType

- The API PduR_<User:LoTp>RxIndication is implemented according to SWS_PduR_00375 ASR 4.1.2 [3]
  - Result type changed to Std_ReturnType

- The API PduR_<User:LoTp>TxConfirmation is implemented according to SWS_PduR_00381 ASR 4.1.2 [3]
  - Result type changed to Std_ReturnType

- The PduR does not provide the return value "BUFREQ_E_BUSY" for the function PduR_<User:LoTp>CopyRxData like specified in Requirement PDUR512 in [1]. The API is implemented according to SWS_PduR_00512 ASR 4.1.1 [2]

- If the PduR_<Lo>StartOfReception is called with an I-PDU ID that is already in process, the PDU Router does not forward the call to the upper module

- In case a transport protocol module reports PduR_<LoTp>TxConfirmation with result other than NTFRSLT_OK the PDUR does not forward the result in the <Up>_TpTxConfirmation to the source upper layer module due to optimization reason.

- The PduR does not support the retry parameter in the PduR_<LoTp>CopyTxData like specified in Requirement PDUR518 in [1]

- State Management: PDUR_REDUCED is not used

- PduR does not return any value if a routing path group is disabled

- PduR does not clear the buffers if a routing path group is disabled

- The header files does not contain software and specification version number

- If the routing path group id does not exist, then the PDUR call the DET instate of return with no action. [PDUR0716]

7.2 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.

7.2.1 General

- Link-time configuration support
- Multiple Configuration support
- 1:N fan-out from the same upper layer PDU
- N:1 fan-in to the same upper layer PDU

8 Glossary and Abbreviations

8.1 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 an 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 8-1 Glossary

8.2 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 8-2 Abbreviations

9 Contact

Visit our website for more information on

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

www.vector.com