VECTOR

MICROSAR vIpc

Technical Reference

Inter Processor Communication

Version 0.1.0

Authors	David Zentner
Status	Not Released

Document Information

History

Author	Date	Version	Remarks
David Zentner	2017-08-01	0.0.0	Creation
David Zentner	2017-08-09	0.1.0	STORYC-1454

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_SWS_DET.pdf	[version]
[2]	AUTOSAR	AUTOSAR_SWS_DEM.pdf	[version]
[3]	AUTOSAR	AUTOSAR_BasicSoftwareModules.pdf	V1.0.0

Scope of the Document

This technical reference describes the general use of the vIpc basis software.

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
0.02.00	Support of segmented and unsegmented transmission and reception.
0.03.00	Support of priority Tx queue.

Table 1-1 Component history

2 Introduction

This document describes the functionality, API and configuration of the AUTOSAR BSW module vIpc.

Supported AUTOSAR Release*:	4	
Supported Configuration Variants:	pre-compile, link-time	
Vendor ID:	VIPC_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	VIPC_MODULE_ID	255 decimal (according to ref. [3])

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

The module vIpc is used for communication between different ECUs via different channels. It establishes a point-to-point connection to its target where multiple connections can share one communication channel. Thus, the connections are multiplexed on one channel. One channel can multiplex up to 256 connections.

A channel is always unidirectional; therefore there is no flow control in vIpc.

vIpc supports communication via bus systems as well as via memory access depending of the configured lower layer.

2.1 Architecture Overview

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

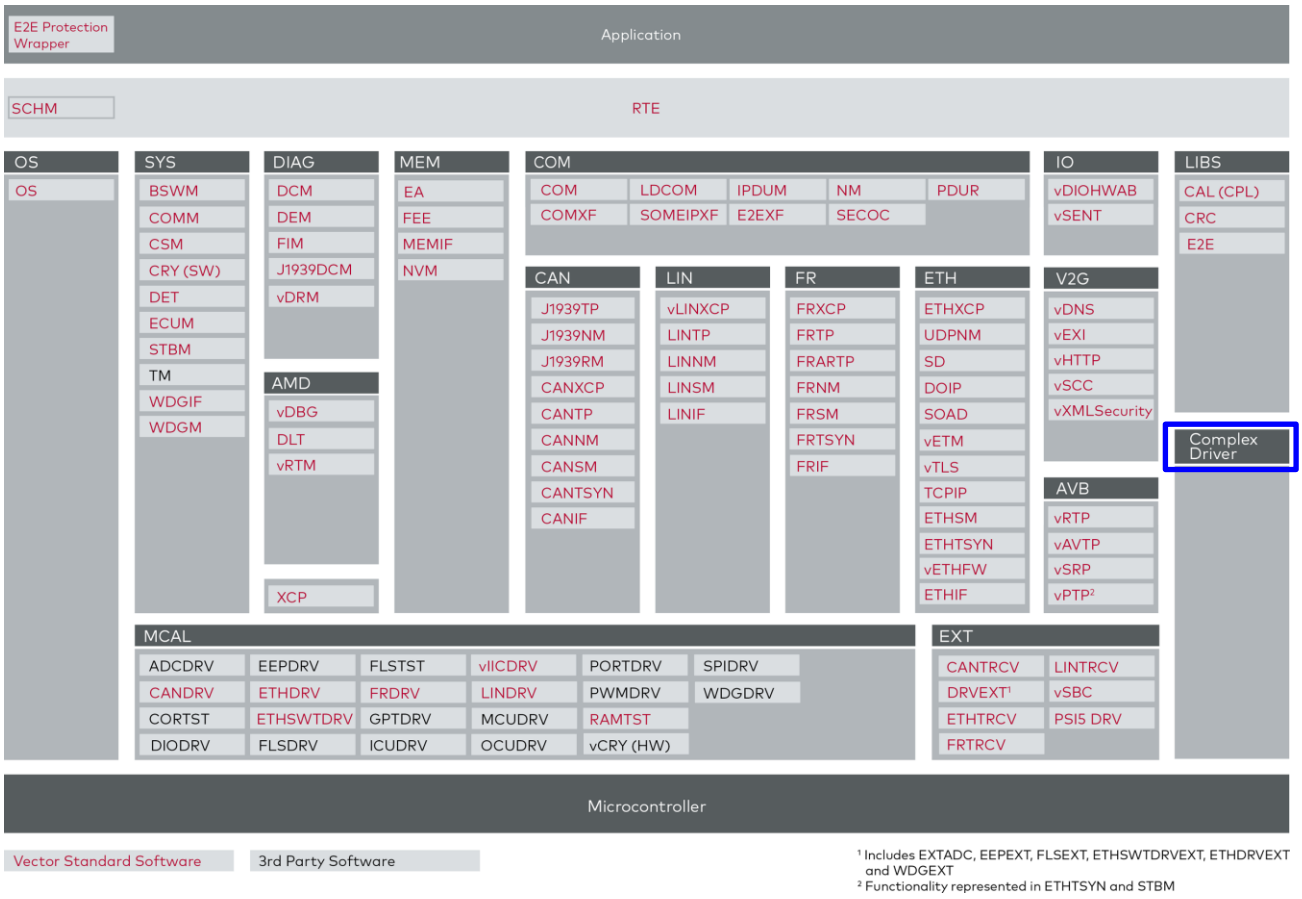

Figure 2-1 AUTOSAR 4.2 Architecture Overview

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

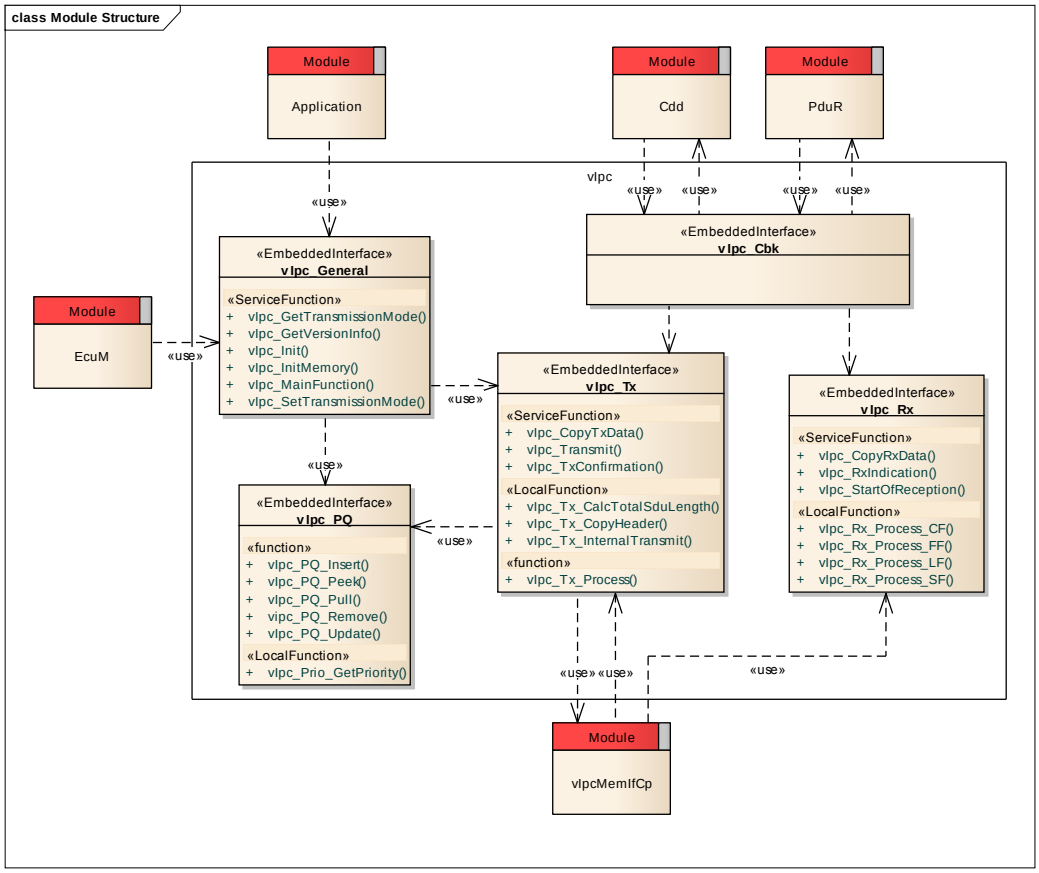

Figure 2-2 Interfaces to adjacent modules of the vIpc

3 Functional Description

3.1 Features

The features listed in the following table cover the complete functionality specified for the vIpc.

3.1.1 Supported Features

The following features are supported:

Supported Features
Support of different UL and LL.
Unsegmented and segmented transmission and reception (TP-Interfaces).
Multiplexing of up to 256 connections on one channel.
Bandwidth control.
Tx queue; FiFo

Table 3-1 Supported features

3.1.2 Deviations

There are no known unsupported features.

Category	Description	Version

Table 3-2 Deviations from standards

3.2 Initialization

The vIpc module is pre-initialized by a call to function vIpc_InitMemory.

The vIpc module is initialized by a call to function vIpc_Init. Initialization of the vIpc must not be done before the used lower layer (e.g. vIpc_MemIf) is initialized.

3.3 States

The vIpc has the following state machine:

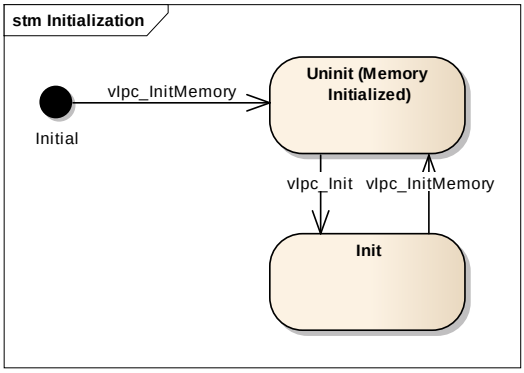

Figure 3-1 Initialization state machine

In state `Uninit` vIpc rejects all transmission requests.

After call to vIpc_Init the vIpc is initialized and all supported features are available.

3.4 Main Functions

The vIpc provides the vIpc_MainFunction which has to be called cyclically.

The main function triggers transmission on every Tx channel where a transmission request is pending.

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 [1], if development error reporting is enabled (i.e. pre-compile parameter `VIPC_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 vIpc Module ID is 255.

The reported service IDs identify the services which are described in 5.1. The following table presents the service IDs and the related services:

Service ID	Service
0x00	vIpc_Init
0x01	vIpc_InitMemory
0x02	vIpc_GetVersionInfo
0x03	vIpc_Transmit
0x04	vIpc_RxIndication
0x05	vIpc_TxConfirmation
0x06	vIpc_CopyRxData
0x07	vIpc_CopyTxData
0x08	vIpc_StartOfReception

Table 3-3 Service IDs

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

Error Code		Description
0x01	VIPC_E_INV_POINTER	The pointer is invalid and points to NULL.
0x02	VIPC_E_INV_CONFIG	The configuration is invalid; the provided PDU length is invalid.
0x03	VIPC_E_INV_CHANNEL	Invalid channel identifier.
0x04	VIPC_E_INV_PHYS_CHANNEL	Invalid physical channel identifier.
0x05	VIPC_E_INV_SDU_ID	The given SDU (connection) ID is invalid.

Error Code		Description
0x06	VIPC_E_INV_PDU_ID	The given PDU (channel) ID is invalid.
0x10	VIPC_E_NOT_INITIALIZED	vIpc is not initialized, yet.
0x11	VIPC_E_ALREADY_INITIALIZED	vIpc is already initialized.

Table 3-4 Errors reported to DET

3.5.2 Production Code Error Reporting

vIpc does not provide Production Code Error Reporting.

3.6 Support of different UL and LL

The vIpc supports the PduR and a CDD as UL. For this, Ipc provides the TP-APIs.

Additionally, the vIpc supports different LL, e.g. the vIpcMemIf. The vIpc also requires the TP-APIs of its LL.

3.7 Unsegmented and segmented transmission and reception

The SDU length of UL (connection) may be larger than the PDU length of the assigned LL (channel), in this case the transmission data in segmented in a FF, optional CFs and a LF.

Otherwise, the transmission data is transmitted in a SF.

In both cases the UL only triggers once the transmission, receives multiple calls to <UL>_CopyTxData and one call to its <UL>_TxConfirmation after the complete PDU is transmitted.

The same goes for the reception path. The receiving UL receives one call to <UL>_StartOfReception, multiple calls to <UL>_CopyRxData and one call to <UL>_RxIndication after complete PDU is received.

3.8 Multiplexing of up to 256 connections on one channel

A vIpc channel is a unidirectional communication path to another ECU which can hold multiple connections. A connection represents one user of the vIpc.

If multiple connections share one channel, the transmission order is managed according to the buffer implementation. Please refer to chapter 3.9.

For each Tx channel there must be an Rx channel on another ECU. The same goes for Tx connections and Rx connections.

The mapping has to be done manually by referencing the same global PDUs on Tx and Rx side. Additionally, the address of each connection has to be set accordingly. A Tx connection is mapped to a Rx connection if the addresses are equal.

If /MICROSAR/vIpc/vIpcConfig/vIpcTxChannel/vIpcTxConnection/**vIpcTxAddress** == /MICROSAR/vIpc/vIpcConfig/vIpcRxChannel/vIpcRxConnection/**vIpcRxAddress.**

Caution
If there are **n** connections on one Tx channel, on the receiving ECU there has to be an Rx channel also with **n** connections.

The mapping of Tx channel to Rx channel and Tx connection to Rx connection has to be done manually.

For example:

ECU_0	ECU_1
> Tx_Channel_0 -> PDU_0	> Rx_Channel_0 -> PDU_0
> Tx_Connection_0	> Rx_Connection_0
> Address = 0	> Address = 1
> Tx_Connection_1	> Rx_Connection_1
> Address = 1	> Address = 0

In this example the following objects are mapped:

> Tx_Channel_0 is mapped to Rx_Channel_0

> Tx_Connection_0 is mapped to Rx_Connection_1

> Tx_Connection_1 is mapped to Rx_Connection_0

3.9 Tx Queue

Each Tx channel has its own queue to buffer transmission requests for each connection. Each connection must only be queued once; as long as a connection is queued other transmission requests of this connection are rejected. A queued connection is removed from the queue, when the transmission of the last PDU segment is confirmed.

There are two queue implementations available:

> FiFo

> Priority based

The queue implementation can be selected with the configuration parameter /MICROSAR/vIpc/vIpcConfig/vIpcTxChannel/vIpcSchedulingAlgorithm.

3.9.1 FiFo

The FiFo queue handles the first occurred transmission request as most important. As soon as the Tx channel becomes ready for next transmission, this transmission request is processed.

3.9.2 Priority based

The priority based queue uses the configured priorities of each Tx Connection to decide in which order the processed.

The priority can be configured with the parameter `/MICROSAR/vIpc/vIpcConfig/vIpcTxChannel/vIpcTxConnection/vIpcPriority`.

3.10 Bandwidth control

vIpc provides the possibility to reduce the bandwidth by configuration of `/MICROSAR/vIpc/vIpcConfig/vIpcTxChannel/vIpcNumSeparationCycles`.

If this parameter is set to 0, a pending transmission is triggered on the lower layer as soon as the current transmission is confirmed. This is the burst mode.

If this parameter is greater than 0, then it specifies the number of vIpc_MainFunctions that have to be executed before the next transmission is triggered.

Caution
The parameter `/MICROSAR/vIpc/vIpcConfig/vIpcTxChannel/vIpcNumSeparationCycles` must be set to greater 0 if the TxConfirmation is called by the LL immediately in context of the vIpc_Transmit.

Otherwise, there could be recursion!

4 Integration

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

4.1 Scope of Delivery

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

4.1.1 Static Files

File Name	Description
vIpc.c	This is the source file of the vIpc. It contains the general APIs like the vIpc_Init and the vIpc_MainFunction.
vIpc.h	This is the header file; it contains general data of the vIpc. This file should be included by vIpc's ULs.
vIpc_Cbk.h	This is the callback header; it contains the declarations of callback function. This file should be included by vIpc's LL.
vIpc_PQ.c	This is the source file containing implementation of the transmission queue.
vIpc_PQ.h	This is the header file for the transmission queue. This should not be included from other modules.
vIpc_Priv.h	This is the private header file for internal data. This should not be included from other modules.
vIpc_Rx.c	This is the source file containing implementation of the reception path.
vIpc_Rx.h	This is the header file for the reception path. This should not be included from other modules.
vIpc_Tx.c	This is the source file containing implementation of the transmission path.
vIpc_Tx.h	This is the header file for the transmission path. This should not be included from other modules.
vIpc_XCfgImpl.h	This is the header file containing the implementation of GenData abstraction. This should not be included from other modules.
vIpc_XCfg.h	This is the header file containing the declaration of GenData abstraction. This should not be included from other modules.

Table 4-1 Static files

4.1.2 Dynamic Files

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

File Name	Description
vIpc_Cfg.h	This is the precompile configuration file containing definitions of global GenData.
vIpc_Lcfg.c	This is the link time configuration file containing definitions of global GenData.
vIpc_Lcfg.h	This is the link time configuration header containing declaration of global GenData.

File Name	Description
vIpc_PBcfg.c	This is the post-build configuration file containing definitions of global GenData.
vIpc_PBcfg.h	This is the post-build configuration header containing declaration of global GenData.
vIpc_Types.h	This is the types header defining all global types.

Table 4-2 Generated files

4.2 Critical Sections

The vIpc module uses the following exclusive area:

> VIPC_EXCLUSIVE_AREA_TX_QUEUE

It is entered for every Tx queue access.

5 API Description

For an interfaces overview please see Figure 2-2.

5.1 Services provided by vIpc

5.1.1 vIpc_InitMemory

Prototype	
`void` **`vIpc_InitMemory`** `( void )`	
Parameter	
`N/A`	N/A
Return code	
`void`	N/A
Functional Description	
This API pre-initializes the vIpc.	
Particularities and Limitations	
> Service ID: see table 'Service IDs'. > This function is synchronous. > This function is reentrant.	
Expected Caller Context	
> This function shall be called on task or interrupt level.	

Table 5-1 vIpc_InitMemory

5.1.2 vIpc_Init

Prototype	
`void` **`vIpc_Init`** `(vIpc_ConfigType * ConfigPtr)`	
Parameter	
`ConfigPtr`	A pointer to the current configuration. If no configuration option exist, it may point to NULL.
Return code	
`void`	N/A
Functional Description	
This function initializes the vIpc module.	
Particularities and Limitations	
> Service ID: see table 'Service IDs'. > This function is synchronous. > This function is non-reentrant.	

Expected Caller Context
> This function can be called on interrupt and task level.

Table 5-2 vIpc_Init

5.1.3 vIpc_GetVersionInfo

Prototype	
`void` **`vIpc_GetVersionInfo`** `(Sdt_VersionInfoType * VersionInfo)`	
Parameter	
`VersionInfo`	A pointer to the current version. This must not point to NULL.
Return code	
`void`	N/A
Functional Description	
This function returns the version information of vIpc.	
Particularities and Limitations	
> Service ID: see table 'Service IDs'. > This function is synchronous. > This function is reentrant.	
Expected Caller Context	
> This function can be called on interrupt and task level.	

Table 5-3 vIpc_GetVersionInfo

5.1.4 vIpc_MainFunction

Prototype	
`void` **`vIpc_MainFunction`** `(void)`	
Parameter	
`void`	N/A
Return code	
`void`	N/A
Functional Description	
This function is executed cyclically and triggers transmission for pending transmission requests.	
Particularities and Limitations	
> Service ID: see table 'Service IDs'. > This function is synchronous. > This function is non-reentrant.	
Expected Caller Context	
> This function can be called on interrupt and task level.	

Table 5-4 vIpc_MainFunction

5.1.5 vIpc_Transmit

Prototype	
`void vIpc_Transmit (PduIdType TxSduId, PduInfoType * PduInfoPtr)`	
Parameter	
`TxSduId`	Identifier of the connection.
`PduInfoPtr`	Pointer to transmission data and its length.
Return code	
`void`	N/A
Functional Description	
This function queues the transmission request. If the bus is idle, the transmission request is forwarded to vIpc's LL.	
Particularities and Limitations	
> Service ID: see table 'Service IDs'. > This function is synchronous. > This function is non-reentrant.	
Expected Caller Context	
> This function can be called on interrupt and task level.	

Table 5-5 vIpc_Transmit

5.2 Services used by vIpc

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

Component	API
DET	Det_ReportError

Table 5-6 Services used by the vIpc

5.3 Callback Functions

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

5.3.1 vIpc_StartOfReception

Prototype	
`BufReq_ReturnType vIpc_StartOfReception(PduIdType RxPduId, PduInfoType * PduInfoPtr, PduLengthType TpSduLength, PduLengthType * BufferSizePtr)`	
Parameter	
`RxPduId`	The identifier of the vIpc channel.

PduInfoPtr	Pointer to the received data and their length. This pointer has to point to valid data and the length must be at least enough to copy the vIpc header (6Byte).
TpSduLength	The length of received data.
BufferSizePtr	Indicates to the LL how much buffer is available on the UL in addition to vIpc header data.
Return code	
BufReq_ReturnType	BUFREQ_OK: Indicates success. BUFREQ_E_NOT_OK: Indicates that copying data failed. BUFREQ_E_BUSY: Indicates that UL has currently not enough buffer.
Functional Description	
This function is called by vIpc's LL to indicate that a PDU is received and ready to be forwarded.	
Particularities and Limitations	
> Service ID: see table 'Service IDs'. > This function is synchronous. > This function is reentrant.	
Expected Caller Context	
> This function can be called on interrupt and task level.	

Table 5-7 vIpc_StartOfReception

5.3.2 vIpc_CopyRxData

Prototype	
`BufReq_ReturnType` **`vIpc_CopyRxData`**`(PduIdType RxPduId, PduInfoType * PduInfoPtr, PduLengthType * BufferSizePtr)`	
Parameter	
RxPduId	The identifier of the vIpc channel.
PduInfoPtr	Pointer to the received data and their length.
BufferSizePtr	Indicates to the LL how much buffer is available on the UL in addition to vIpc header data.
Return code	
BufReq_ReturnType	BUFREQ_OK: Indicates success. BUFREQ_E_NOT_OK: Indicates that copying data failed. BUFREQ_E_BUSY: Indicates that UL has currently not enough buffer.
Functional Description	
This function is called by vIpc's LL if more data of the current PDU is received.	
Particularities and Limitations	
> Service ID: see table 'Service IDs'. > This function is synchronous. > This function is reentrant.	

Expected Caller Context
> This function can be called on interrupt and task level.

Table 5-8 vIpc_CopyRxData

5.3.3 vIpc_RxIndication

Prototype	
`void` **`vIpc_RxIndication`**`(PduIdType RxPduId, Std_ReturnType Result)`	
Parameter	
`RxPduId`	The identifier of the vIpc channel.
`Result`	`E_OK`: Indicates success. `E_NOT_OK`: Indicates error during reception sequence.
Return code	
`void`	N/A
Functional Description	
This function is called by vIpc's LL after a successful reception of a PDU.	
Particularities and Limitations	
> Service ID: see table 'Service IDs'. > This function is synchronous. > This function is reentrant.	
Expected Caller Context	
> This function can be called on interrupt and task level.	

Table 5-9 vIpc_RxIndication

5.3.4 vIpc_CopyTxData

Prototype	
`BufReq_ReturnType` **`vIpc_CopyTxData`**`(PduIdType TxPduId, PduInfoType * PduInfoPtr, RetryInfoType * RetryPtr, PduLengthType * AvailableDataPtr)`	
Parameter	
`RxPduId`	The identifier of the vIpc channel.
`PduInfoPtr`	Pointer to the Tx buffer of vIpc's LL and its length.
`RetryPtr`	Indicates if previously transmitted data has to be re-transmitted. This parameter is not supported by vIpc.
`AvailableDataPtr`	Indicates to the LL how much buffer is available on the UL in addition to vIpc header data.
Return code	
`BufReq_ReturnType`	`BUFREQ_OK`: Indicates success. `BUFREQ_E_NOT_OK`: Indicates that copying data failed. `BUFREQ_E_BUSY`: Indicates that UL has currently not enough buffer.

Functional Description
This function is called by vIpc's LL if more data of the current PDU can be transmitted.
Particularities and Limitations
> Service ID: see table 'Service IDs'. > This function is synchronous. > This function is reentrant.
Expected Caller Context
> This function can be called on interrupt and task level.

Table 5-10 vIpc_CopyTxData

5.3.5 vIpc_TxConfirmation

Prototype	
`void` **`vIpc_TxConfirmation`**`(PduIdType TxPduId, Std_ReturnType Result)`	
Parameter	
`TxPduId`	The identifier of the vIpc channel.
`Result`	`E_OK`: Indicates success. `E_NOT_OK`: Indicates error during transmission sequence.
Return code	
`void`	N/A
Functional Description	
This function is called by vIpc's LL to confirm the successful transmission of a PDU.	
Particularities and Limitations	
> Service ID: see table 'Service IDs'. > This function is synchronous. > This function is reentrant.	
Expected Caller Context	
> This function can be called on interrupt and task level.	

Table 5-11 vIpc_TxConfirmation

6 Configuration

6.1 Configuration Variants

The vIpc supports the configuration variants

- VARIANT-PRE-COMPILE
- VARIANT-LINK-TIME

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

7 Glossary and Abbreviations

7.1 Glossary

Term	Description
EAD	Embedded Architecture Designer; generation tool for MICROSAR components
SF	Single Frame
FF	First Frame
CF	Consecutive Frame
LF	Last Frame
FiFo	First In First Out
UL	Upper Layer
LL	Lower Layer

Table 7-1 Glossary

7.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
DEM	Diagnostic Event Manager
DET	Development Error Tracer
EAD	Embedded Architecture Designer
ECU	Electronic Control Unit
HIS	Hersteller Initiative Software
ISR	Interrupt Service Routine
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
PPORT	Provide Port
RPORT	Require Port
RTE	Runtime Environment
SRS	Software Requirement Specification
SWC	Software Component
SWS	Software Specification

Table 7-2 Abbreviations

8 Contact

Visit our website for more information on

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

www.vector.com