VECTOR

MICROSAR vIpcMemCp

Technical Reference

Inter Processor Communication via Shared Memory

Version 1.1.0

Authors	Benjamin Seifert
Status	Released

Document Information

History

Author	Date	Version	Remarks
Benjamin Seifert	2017-08-04	1.0.0	Initial version
Benjamin Seifert	2017-08-08	1.1.0	Updated API descriptions Some editorial changes

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_BasicSoftwareModules.pdf	V1.0.0
[2]	AUTOSAR	AUTOSAR_SWS_DevelopmentErrorTracer.pdf	V3.2.0
[3]	Vector	TechnicalReference_PostBuildLoadable.pdf	see delivery

Caution
We have configured the programs in accordance with your specifications in the questionnaire. Whereas the programs do support other configurations than the one specified in your questionnaire, Vector´s release of the programs delivered to your company is expressly restricted to the configuration you have specified in the questionnaire.

Contents

Illustrations

Tables

1 Introduction

This document describes the functionality, API and configuration of the MICROSAR BSW module vIpcMemCp.

Supported AUTOSAR Release:	4	
Supported Configuration Variants:	pre-compile, post-build loadable	
Vendor ID:	VIPCMEMCP_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	VIPCMEMCP _MODULE_ID	255 decimal (according to ref. [1])

The vIpcMemCp (Vector interprocessor communication via shared memory for classic AUTOSAR) module is used to transfer data between 2 or more instances via shared memory.

The instances are usually controlled by different CPU cores. These may be controlled by different Operating Systems and may in some circumstances even have different architectures. The requirement is that they operate on the same memory.

Multiple channels are supported. Each channel is unidirectional. The channels are fully independent from each other.

1.1 Architecture Overview

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

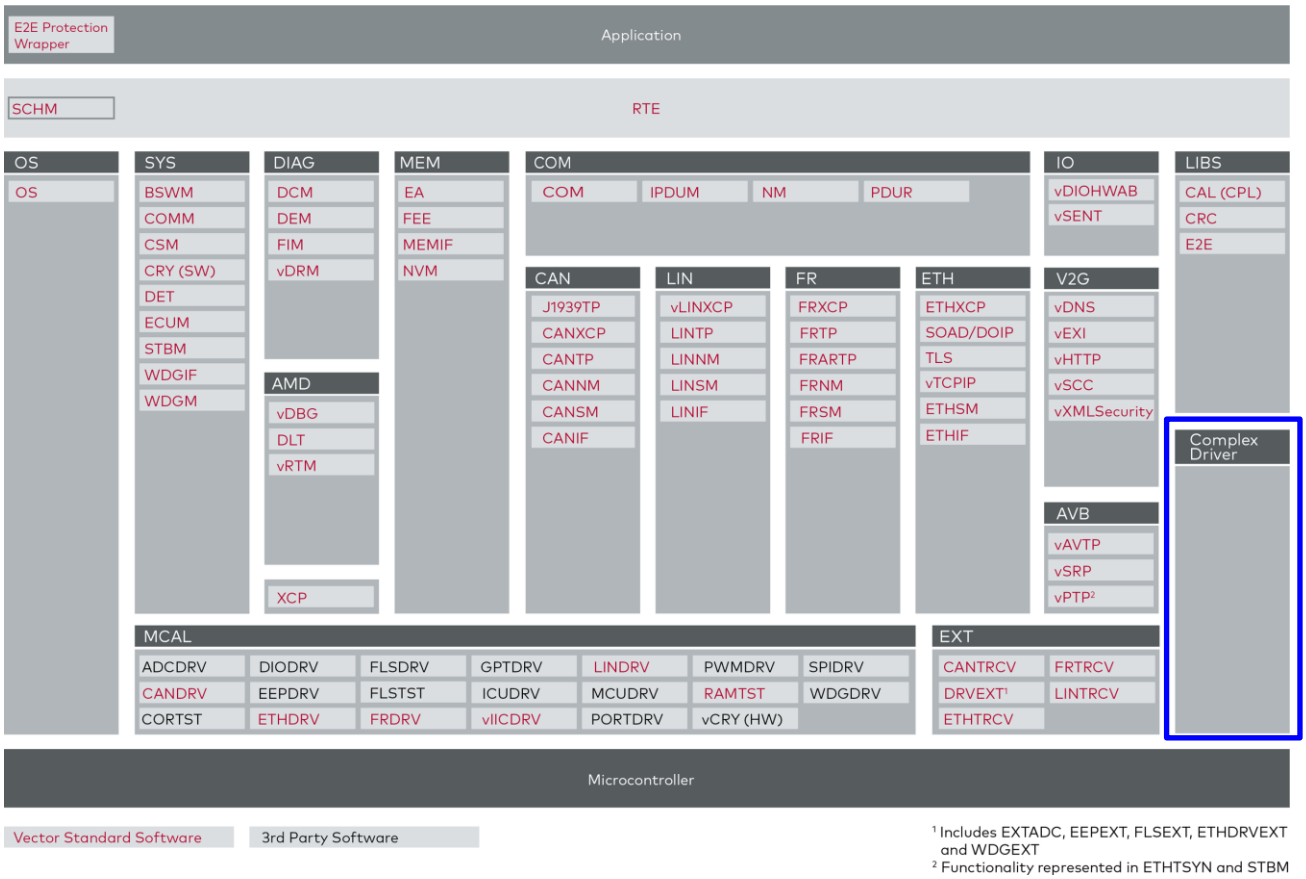

Figure 1-1 AUTOSAR 4.x Architecture Overview

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

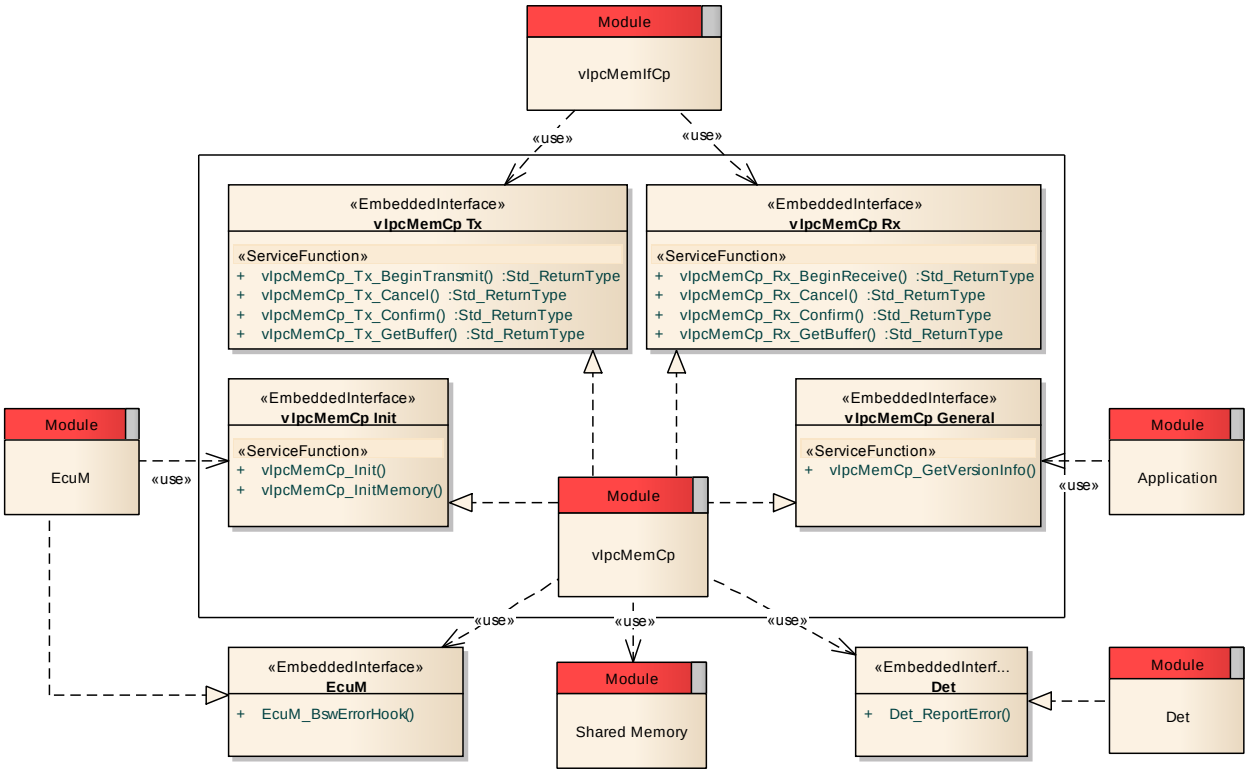

Figure 1-2 Interfaces to adjacent modules of the vIpcMemCp

2 Functional Description

2.1 Initialization

The vIpcMemCp module is pre-initialized by a call to function `vIpcMemCp_InitMemory()`.

The vIpcMemCp module is initialized by a call to function `vIpcMemCp_Init()`.

Caution
The `vIpcMemCp_Init()` function only initializes the configuration data of the current instance. Before the first transmission between two instances both sides has to be initialized first!

2.2 States

The vIpcMemCp does not use any state machines.

2.3 Main Functions

The vIpcMemCp does not have any main functions.

2.4 Error Handling

2.4.1 Development Error Reporting

By default, development errors are reported to the DET using the service `Det_ReportError()` as specified in [2], if development error reporting is enabled (i.e. pre-compile parameter `VIPCMEMCP_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 vIpcMemCp ID is 255.

The reported service IDs identify the services which are described in 4.2. The following table presents the service IDs and the related services:

Service ID		Service
0x00	VIPCMEMCP_SID_INIT	vIpcMemCp_Init
0x01	VIPCMEMCP_SID_INIT_MEMORY	vIpcMemCp_InitMemory
0x02	VIPCMEMCP_SID_GET_VERSION_INFO	vIpcMemCp_GetVersionInfo
0x03	VIPCMEMCP_SID_TX_BEGINTRANSMIT	vIpcMemCp_Tx_BeginTransmit
0x04	VIPCMEMCP_SID_TX_GETBUFFER	vIpcMemCp_Tx_GetBuffer
0x05	VIPCMEMCP_SID_TX_CONFIRM	vIpcMemCp_Tx_Confirm
0x06	VIPCMEMCP_SID_TX_CANCEL	vIpcMemCp_Tx_Cancel
0x07	VIPCMEMCP_SID_RX_BEGINRECEIVE	vIpcMemCp_Rx_BeginReceive

Service ID		Service
0x08	VIPCMEMCP_SID_RX_GETBUFFER	vIpcMemCp_Rx_GetBuffer
0x09	VIPCMEMCP_SID_RX_CONFIRM	vIpcMemCp_Rx_Confirm
0x0A	VIPCMEMCP_SID_RX_CANCEL	vIpcMemCp_Rx_Cancel

Table 2-1 Service IDs

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

Error Code		Description
0x01	VIPCMEMCP_E_INV_PTR	Invalid pointer passed to function.
0x02	VIPCMEMCP_E_PARAM	Invalid parameter passed to function.
0x03	VIPCMEMCP_E_UNINIT	API service used without module initialization.
0x04	VIPCMEMCP_E_INCONSISTENT_SIZE	Size during finalizing is inconsistent with previously used data.
0x05	VIPCMEMCP_E_TX_ALREADY_PENDING	Transmit is not possible due to another one still pending.
0x06	VIPCMEMCP_E_TX_NOT_PENDING	API service is used before transmission was started.
0x07	VIPCMEMCP_E_RX_ALREADY_PENDING	Receive is not possible due to another one still pending.
0x08	VIPCMEMCP_E_RX_NOT_PENDING	API service is used before reception was started.

Table 2-2 Errors reported to DET

2.4.2 Production Code Error Reporting

Production code error reporting is not supported in the current version of the vIpcMemCp.

2.5 Transmission (TX)

See the following sequence diagram for a complete transmission.

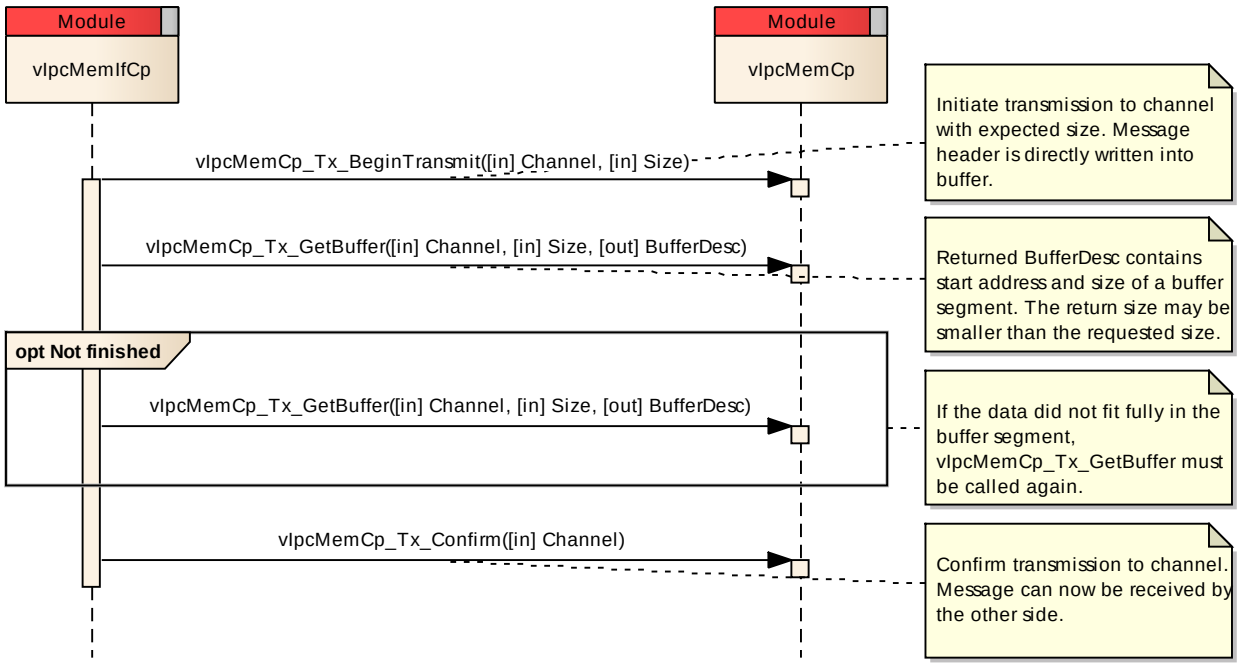

Figure 2-1 Transmission Sequence

2.6 Reception (RX)

See the following sequence diagram for a complete reception.

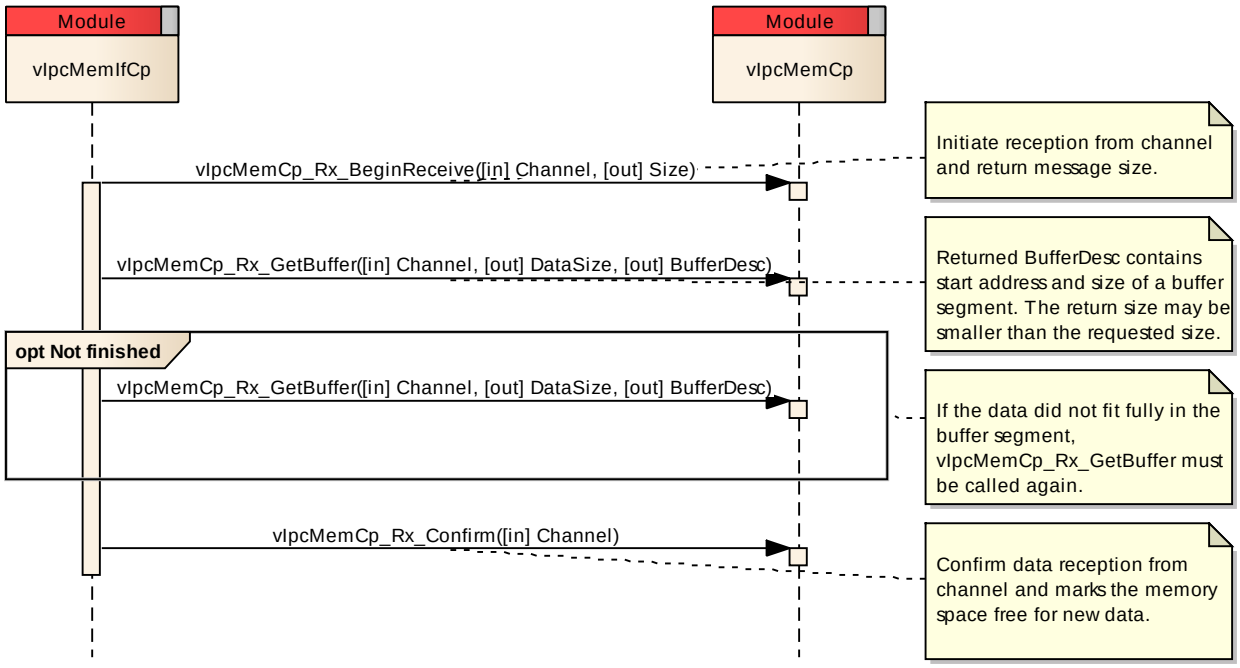

Figure 2-2 Reception Sequence

3 Integration

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

3.1 Configuration of Transmission

To initiate a transmission between two independent instances both sides need to have one TX respectively one RX channel configured with the same start address and size.

For example a transmission from Instance A (TX) to Instance B (RX):

Instance A	Instance B	Value
vIpcMemCp**Tx**Channel	vIpcMemCp**Rx**Channel	Channel0
vIpcMemCp**Tx**ChannelId	vIpcMemCp**Rx**ChannelId	0
vIpcMemCp**Tx**StartAddress	vIpcMemCp**Rx**StartAddress	_StartAddressLabelCh0
vIpcMemCp**Tx**BufferSize	vIpcMemCp**Rx**BufferSize	4096

Table 3-1 Example configuration of transmission

Note
The start address parameters can be specified either as a valid decimal (1024) or hexadecimal (0x4000) value, or as a valid linker symbol.

Caution
The application using a TX or RX channel needs to have write or read access rights to the shared memory buffer.

3.2 Scope of Delivery

The delivery of the vIpcMemCp contains the files which are described in the chapters 3.2.1 and 3.2.2:

3.2.1 Static Files

File Name	Description
`vIpcMemCp.c`	Contains the implementation of the external API functions.
`vIpcMemCp.h`	Contains the declaration of the external API functions.
`vIpcMemCp_Int.h`	Contains the declaration and definition of general internal functions.
`vIpcMemCp_RxInt.h`	Contains the declaration and definition of TX internal functions.
`vIpcMemCp_TxInt.h`	Contains the declaration and definition of RX internal functions.
`vIpcMemCp_Types.h`	Contains the type definitions of the module.

Table 3-2 Static files

3.2.2 Dynamic Files

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

File Name	Description
vIpcMemCp_Cfg.c	Contains the pre-compile configuration of this module.
vIpcMemCp_Cfg.h	Contains the pre-compile configuration data declarations.
vIpcMemCp_PBcfg.c	Contains the post-build configuration of this module.
vIpcMemCp_PBcfg.h	Contains the post-build configuration data declarations.

Table 3-3 Generated files

4 API Description

For an interfaces overview please see Figure 1-2.

4.1 Type Definitions

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

Type Name	C-Type	Description	Value Range
`vIpcMemCp_ChannelIndexType`	`uint8_least`	Type to address a channel.	0..255

Table 4-1 Type definitions

vIpcMemCp_BufferDescriptionType

This structure contains the description of a single continuous data block.

Struct Element Name	C-Type	Description	Value Range
`Ptr`	`uint8*`	Pointer to the start address of the data block.	0..4294967295
`Size`	`uint32`	Data block size.	0..4294967295

Table 4-2 vIpcMemCp_BufferDescriptionType

4.2 Services provided by vIpcMemCp

4.2.1 vIpcMemCp_InitMemory

Prototype	
`void` **`vIpcMemCp_InitMemory`** `(void)`	
Parameter	
void	none
Return code	
void	none
Functional Description	
Function for *_INIT_*-variable initialization.	
Particularities and Limitations	
Module is uninitialized. Service to initialize module global variables at power up. This function initializes the variables in *_INIT_* sections. Used in case they are not initialized by the startup code.	
Call context	
> ANY > This function is Synchronous > This function is Non-Reentrant	

Table 4-3 vIpcMemCp_InitMemory

4.2.2 vIpcMemCp_Init

Prototype	
`void vIpcMemCp_Init (const vIpcMemCp_ConfigType *ConfigPtr)`	
Parameter	
ConfigPtr [in]	Pointer to the Post-Build loadable configuration
Return code	
void	none
Functional Description	
Initialization function.	
Particularities and Limitations	
Module is uninitialized. This function initializes the module vIpcMemCp. It initializes all intern variables.	
Call context	
> ANY > This function is Synchronous > This function is Non-Reentrant	

Table 4-4 vIpcMemCp_Init

4.2.3 vIpcMemCp_GetVersionInfo

Prototype	
`void vIpcMemCp_GetVersionInfo (Std_VersionInfoType *VersionInfo)`	
Parameter	
VersionInfo [out]	Pointer to where to store the version information. Parameter must not be NULL.
Return code	
void	none
Functional Description	
Returns the version information.	
Particularities and Limitations	
This function returns version information, vendor ID and AUTOSAR module ID of the component. Configuration Variant(s): VIPCMEMCP_VERSION_INFO_API	
Call context	
> ANY > This function is Synchronous > This function is Reentrant	

Table 4-5 vIpcMemCp_GetVersionInfo

4.2.4 vIpcMemCp_Tx_BeginTransmit

Prototype	
`Std_ReturnType` **`vIpcMemCp_Tx_BeginTransmit`** `(vIpcMemCp_ChannelIndexType ChannelNr, uint32 DataSize)`	
Parameter	
ChannelNr [in]	The channel index to identify the used channel.
DataSize [in]	Length of the user's data in bytes.
Return code	
Std_ReturnType	E_OK Transmission request accepted. E_NOT_OK Transmission request failed.
Functional Description	
Function to initiate a sending transmission.	
Particularities and Limitations	
This functions checks if sufficient free memory is available and prepares the transmission.	
Call context	
> ANY > This function is Synchronous > This function is Reentrant for different channels	

Table 4-6 vIpcMemCp_Tx_BeginTransmit

4.2.5 vIpcMemCp_Tx_GetBuffer

Prototype	
`Std_ReturnType` **`vIpcMemCp_Tx_GetBuffer`** `(vIpcMemCp_ChannelIndexType ChannelNr, uint32 DataSize, vIpcMemCp_BufferDescriptionType *BufferDesc)`	
Parameter	
ChannelNr [in]	The channel index to identify the used channel.
DataSize [in]	Length of the user's data in bytes.
BufferDesc [out]	Buffer description to be used for data copy.
Return code	
Std_ReturnType	E_OK Buffer request was successful. E_NOT_OK Buffer request has failed.
Functional Description	
Function to retrieve a buffer description of a single continuous data block.	
Particularities and Limitations	
The resulted buffer may be smaller than requested data size. In this case a further call of the same function is expected.	
Call context	
> ANY > This function is Synchronous	

> This function is Reentrant for different channels

Table 4-7 vIpcMemCp_Tx_GetBuffer

4.2.6 vIpcMemCp_Tx_Confirm

Prototype	
`Std_ReturnType vIpcMemCp_Tx_Confirm (vIpcMemCp_ChannelIndexType ChannelNr)`	
Parameter	
ChannelNr [in]	The channel index to identify the used channel.
Return code	
Std_ReturnType	E_OK Confirmation was successful. E_NOT_OK Confirmation has failed.
Functional Description	
Function marks the active transmission as finished.	
Particularities and Limitations	
After the call the message will be ready to be read by the receiver.	
Call context	
> ANY > This function is Synchronous > This function is Reentrant for different channels	

Table 4-8 vIpcMemCp_Tx_Confirm

4.2.7 vIpcMemCp_Tx_Cancel

Prototype	
`Std_ReturnType vIpcMemCp_Tx_Cancel (vIpcMemCp_ChannelIndexType ChannelNr)`	
Parameter	
ChannelNr [in]	The channel index to identify the used channel.
Return code	
Std_ReturnType	E_OK Cancel was successful. E_NOT_OK Cancel has failed.
Functional Description	
Function cancels the currently active transmission.	
Particularities and Limitations	
Previously copied data is discarded.	
Call context	
> ANY > This function is Synchronous > This function is Reentrant for different channels	

Table 4-9 vIpcMemCp_Tx_Cancel

4.2.8 vIpcMemCp_Rx_BeginReceive

Prototype	
`Std_ReturnType` **`vIpcMemCp_Rx_BeginReceive`** `(vIpcMemCp_ChannelIndexType ChannelNr, uint32 *DataSize)`	
Parameter	
ChannelNr [in]	The channel index to identify the used channel.
DataSize [out]	Length of the user's data in bytes.
Return code	
Std_ReturnType	E_OK Reception request accepted. E_NOT_OK Reception request failed.
Functional Description	
Function to initiate a reception.	
Particularities and Limitations	
This functions checks if messages are available for reception.	
Call context	
> ANY > This function is Synchronous > This function is Reentrant for different channels	

Table 4-10 vIpcMemCp_Rx_BeginReceive

4.2.9 vIpcMemCp_Rx_GetBuffer

Prototype	
`Std_ReturnType` **`vIpcMemCp_Rx_GetBuffer`** `(vIpcMemCp_ChannelIndexType ChannelNr, uint32 DataSize, vIpcMemCp_BufferDescriptionType *BufferDesc)`	
Parameter	
ChannelNr [in]	The channel index to identify the used channel.
DataSize [in]	Length of the user's data in bytes.
BufferDesc [out]	Buffer description to be used for data copy.
Return code	
Std_ReturnType	E_OK Buffer request was successful. E_NOT_OK Buffer request has failed.
Functional Description	
Function to retrieve a buffer description of a single continuous data block.	
Particularities and Limitations	
The resulted buffer may be smaller than requested data size. In this case a further call of the same function is expected.	
Call context	
> ANY > This function is Synchronous	

> This function is Reentrant for different channels

Table 4-11 vIpcMemCp_Rx_GetBuffer

4.2.10 vIpcMemCp_Rx_Confirm

Prototype	
`Std_ReturnType vIpcMemCp_Rx_Confirm (vIpcMemCp_ChannelIndexType ChannelNr)`	
Parameter	
ChannelNr [in]	The channel index to identify the used channel.
DataSize [in]	Length of the user's data in bytes.
Return code	
Std_ReturnType	E_OK Confirmation was successful. E_NOT_OK Confirmation has failed.
Functional Description	
Function marks the active transmission as finished.	
Particularities and Limitations	
After the call the message space is freed and can be used to transmit further data.	
Call context	
> ANY > This function is Synchronous > This function is Reentrant for different channels	

Table 4-12 vIpcMemCp_Rx_Confirm

4.2.11 vIpcMemCp_Rx_Cancel

Prototype	
`Std_ReturnType vIpcMemCp_Rx_Cancel (vIpcMemCp_ChannelIndexType ChannelNr)`	
Parameter	
ChannelNr [in]	The channel index to identify the used channel.
Return code	
Std_ReturnType	E_OK Cancel was successful. E_NOT_OK Cancel has failed.
Functional Description	
Function cancels the currently active reception.	
Particularities and Limitations	
The unreceived message is discarded.	
Call context	
> ANY > This function is Synchronous > This function is Reentrant for different channels	

Table 4-13 vIpcMemCp_Rx_Cancel

4.3 Services used by vIpcMemCp

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

Component	API
Det	`Det_ReportError`
EcuM	`EcuM_BswErrorHook`

Table 4-14 Services used by the vIpcMemCp

5 Configuration

5.1 Configuration Variants

The vIpcMemCp supports the configuration variants

- VARIANT-PRE-COMPILE
- VARIANT-POST-BUILD-LOADABLE

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

5.2 Configuration of Post-Build

The configuration of post-build loadable is described in see [3].

This component uses a static RAM management which differs from the concept described in the mentioned post-build documentation.

6 Glossary and Abbreviations

6.1 Glossary

Term	Description
BSWMD	The BSWMD is a formal notation of all information belonging to a certain BSW artifact (BSW module or BSW cluster) in addition to the implementation of that artifact.
Buffer	A buffer in a memory area normally in the RAM, that the application has reserved for data storage.
Channel	A channel defines the assignment (1:1) between a physical communication interface and a physical layer, on which different modules are connected to. 1 channel consists of 1..X network(s).
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.
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.
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 reconfiguration possible, the reconfigurable parameters shall be stored at a known memory location of ECU storage area.

Table 6-1 Glossary

6.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
CAN	Controller Area Network protocol originally defined for use as a communication network for control applications in vehicles.
CPU	Central Processing Unit
DET	Development Error Tracer
ECU	Electronic Control Unit
HIS	Hersteller Initiative Software
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
ID	Identifier (e.g. Identifier of a CAN message)
RAM	Random Access Memory
SWS	Software Specification

Table 6-2 Abbreviations

7 Contact

Visit our website for more information on

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

www.vector.com