VECTOR

MICROSAR SAE J1939 Request Manager

Technical Reference

Version 2.0.1

Authors	Simon Gutjahr, Martin Schlodder, Thomas Albrecht
Status	Released

Document Information

History

Author	Date	Version	Remarks
Thomas Albrecht	2013-09-26	0.1.0	Created initial version
Martin Schlodder	2014-06-20	0.2.0	Updated architecture overview
Martin Schlodder	2015-02-20	0.2.1	Renamed document
Martin Schlodder	2015-09-08	1.0.0	First released version
Martin Schlodder	2016-01-12	1.0.1	Improved description of data consistency
Martin Schlodder	2016-01-13	1.1.0	Support of Request2
Martin Schlodder	2016-03-14	1.2.0	Support extended identifier bytes in API
Martin Schlodder	2016-05-09	1.2.1	Variant handling supported
Simon Gutjahr	2016-07-13	1.3.0	Support of runtime errors, support of separate Request2 queue
Simon Gutjahr	2017-04-27	2.0.0	Support of NameManagement PGN (0x9300)
Simon Gutjahr	2017-07-27	2.0.1	Renamed exclusive areas

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_SWS_SAEJ1939RequestManager.pdf	4.2.1
[2]	AUTOSAR	AUTOSAR_SWS_DefaultErrorTracer.pdf	4.2.1
[3]	AUTOSAR	AUTOSAR_SWS_BSWGeneral.pdf	4.2.1
[4]	AUTOSAR	AUTOSAR_TR_BSWModuleList.pdf	4.2.1
[5]	Vector	TechnicalReference_PostBuildLoadable.pdf	1.0.0
[6]	SAE J1939	J1939-21_2015-04.pdf	APR2015

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
[0.1.0]	Initial Version (BETA)
[0.2.0]	Added DET reporting and J1939Rm_GetVersionInfo
[0.3.0]	Added support for post-build configuration and COM users
[0.4.0]	Improved configuration of module initialization in EcuM/BswM
[0.5.0]	Added support for RTE users
[1.0.0]	Support for ISOBUS Acknowledgement added; component released
[1.1.0]	Support for Request2 and multiplexed COM I-PDUs added
[2.0.0]	Support Request2 for CDD and RTE users
[2.1.0]	Added support for variant handling
[2.2.0]	Distinguation of development and runtime errors, separate RQST2 queue

Table 1-1 Component History

2 Introduction

This document describes the functionality, API and configuration of the AUTOSAR BSW module J1939Rm as specified in [1].

Supported AUTOSAR Release:	4	
Supported Configuration Variants:	pre-compile, post-build-loadable, post-build-selectable	
Vendor ID:	J1939RM_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	J1939RM_MODULE_ID	59 decimal (according to [4])

The MICROSAR SAE J1939 Request Manager implements the request/response behavior defined by the SAE in the document J1939-21, including transmission and reception of the Request, the Request2, and the Acknowledgement message. It also supports the Acknowledgement message layout according to ISO 11783-3, which differs in the usage of the destination address.

In the AUTOSAR architecture, the SAE J1939 Request Manager interacts with the PduR to transmit and receive the Request and Acknowledgement messages, and with several upper layers. The J1939Nm is only notified of requests for the AddressClaimed and the NameManagement message. The J1939Dcm is notified of requests for diagnostic messages, and it triggers the J1939Rm to send Acknowledgement in response. COM is triggered to send its messages on request. But only software components using the RTE and CDDs have full access to the functionality of J1939Rm.

J1939Rm uses the Meta Data support introduced with AUTOSAR 4.1.1 to reduce the number of PDUs to one of each type per channel and direction.

2.1 Architecture Overview

The following figure highlights the position of the J1939Rm in the AUTOSAR architecture.

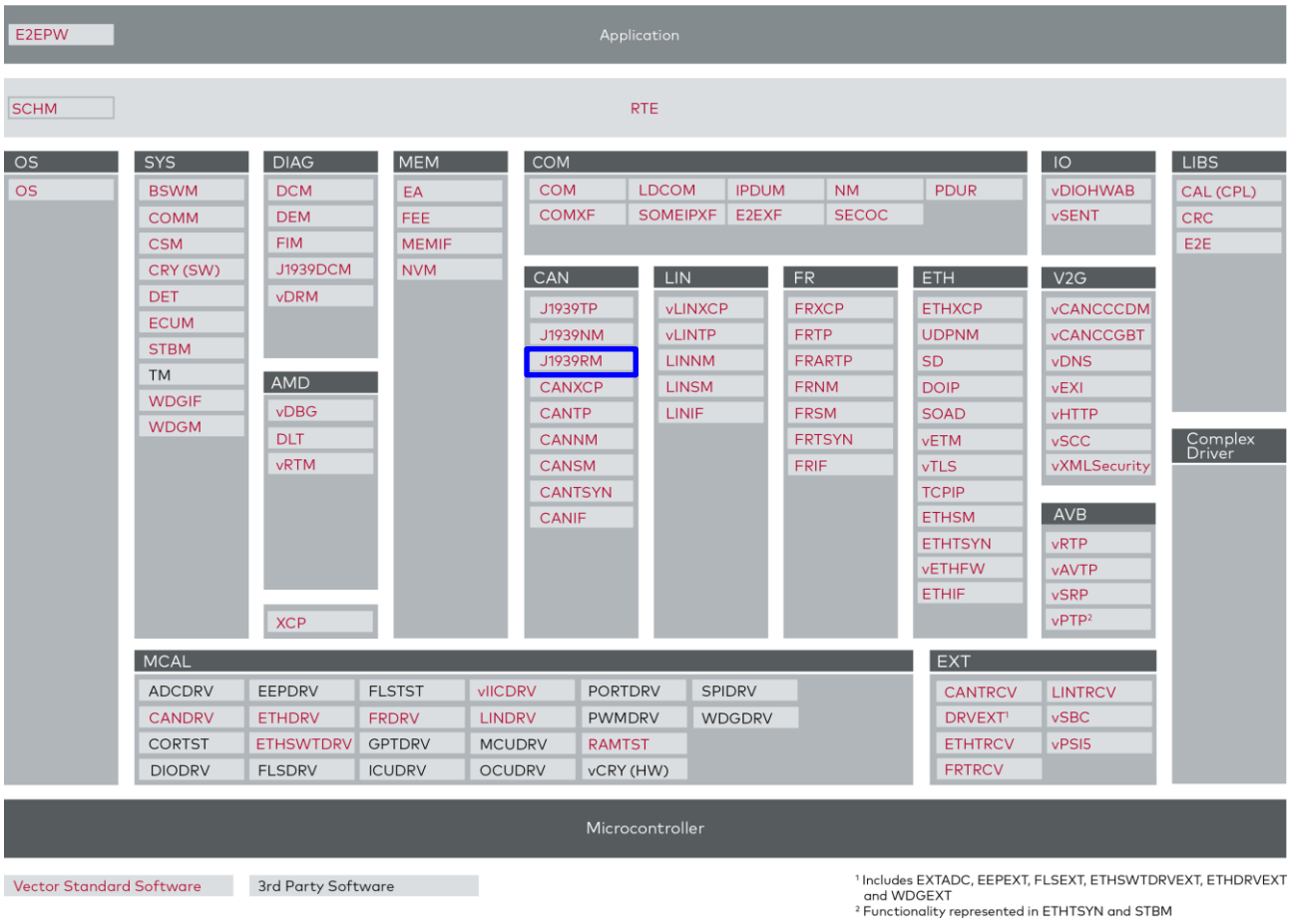

Figure 2-1 AUTOSAR 4.2 Architecture Overview

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 J1939Rm are listed in section 5.6 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 J1939Rm.

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 J1939Rm 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
Initialization and shutdown of the module
Reception and queued transmission of the Request message
Reception and queued transmission of the Acknowledgement message
Interaction with J1939Nm, J1939Dcm, and CDDs
Interaction with COM
Service port interface for software components
Meta data handling

Table 3-1 Supported AUTOSAR standard conform features

3.1.1 Deviations from AUTOSAR 4.2.1

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

Not Supported AUTOSAR Standard Conform Features
Timeout supervision of transmitted Request messages

Table 3-2 Not supported AUTOSAR standard conform features

3.1.1.1 Timeout Supervision of Transmitted Request Messages

Timeout supervision is defined by AUTOSAR to check whether a transmitted request has been answered in time according to [6].

Affected AUTOSAR specification items: SWS_J1939Rm_00017, SWS_J1939Rm_00024, SWS_J1939Rm_00029, SWS_J1939Rm_00030, SWS_J1939Rm_00055, SWS_J1939Rm_00065 , SWS_J1939Rm_00069, SWS_J1939Rm_00075, SWS_J1939Rm_00087, ECUC_J1939Rm_00008, ECUC_J1939Rm_00031, ECUC_J1939Rm_00058

3.1.2 Additions / Extensions

The following features are provided beyond the AUTOSAR standard:

Features Provided Beyond The AUTOSAR Standard
Extended error reporting
Acknowledgement according to ISO 11783-3
Reception and queued transmission of the Request2 message and extended identifier bytes

Table 3-3 Features provided beyond the AUTOSAR standard

3.1.2.1 Extended Error Reporting

The J1939Rm reports additional development errors that are not specified by AUTOSAR. See Table 3-5 in section 3.5.1.

3.1.2.2 Acknowledgement According to ISO 11783-3

The J1939Rm can send the Acknowledgement message with the DA set to the requesting node, as specified by ISO 11783-3.

3.1.2.3 Request2 and Extended Identifier Bytes

The J1939Rm supports reception and transmission of the Request2 and the Acknowledgement message with extended identifier bytes. J1939Rm supports reception and transmission of extended identifier bytes by SW-Cs and CDDs, and can trigger multiplexed ComIPdus depending on the multiplexor values provided as extended identifier bytes with the Request2 message.

3.2 Initialization

The J1939Rm uses a global state (J1939Rm_ModuleInitialized) to determine whether the module is initialized and operational. This state is initially set to J1939RM_UNINIT. If initialization by startup code is not supported, the initialization routine should call J1939Rm_InitMemory() to set the global state to J1939RM_UNINIT.

By calling J1939Rm_Init(), the J1939Rm module is set to the state J1939RM_INIT, and internal states are set to their initial states. The module is now operational.

To stop the J1939Rm module, J1939Rm_DeInit() may be called, which sets the global state to J1939RM_UNINIT again.

3.3 States

The J1939Rm module has a global state, and separate states for each Tx PDU and each Tx Queue (Request, Request2 and Acknowledgement).

3.3.1 Global State

The global state is switched by the services J1939Rm_InitMemory(), J1939Rm_Init(), and J1939Rm_DeInit().

In the state J1939RM_UNINIT, all services of J1939Rm return immediately, reporting J1939RM_E_UNINIT to the DET if enabled. If they have a return value, an error (typically E_NOT_OK) is returned.

In the state J1939RM_INIT, services are operational.

3.3.2 Tx PDU State

Each transmitted PDU has its own state to ensure that a value provided to CanIf is not overwritten with a new one before the CanIf was able to transmit it on the CAN bus. This state is protected by an exclusive area.

The Tx PDU state depends on the J1939Rm_TxConfirmation() being called after each call to PduR_J1939RmTransmit(). Because this is not always the case, the J1939Rm monitors this state and resets it after a timeout configurable via “Tx Confirmation Timeout”, assuming the PDU was not transmitted.

3.3.3 Tx Queue States

Depending on configuration, J1939Rm maintains separate transmit queues for Request, Request2 and Acknowledgement messages for each channel. Each of these queues has a separate state machine that ensures that the read and write indices and the stored messages will not be corrupted upon concurrent access to the queue.

3.3.4 Node State

J1939Rm maintains a separate state for each Node to track whether the node is online or offline. The state can be switched by the service J1939Rm_SetState().

3.4 Main Function

The J1939Rm_MainFunction() is used by the J1939Rm module to supervise the Tx confirmation timeout, and to trigger transmission of queued messages when this timeout occurred. It is important that this main function is called with the timing configured via “Main Function Period”.

Caution

To ensure data consistency, it is essential that the main function never interrupts itself, and does not interrupt any other function more than once, especially the calls to J1939Rm_SendRequest and J1939Rm_SendAck.

In other words, no task that might block the parts of the application that interact with the J1939Rm or other modules of the BSW should take as long as one main function cycle.

3.5 Error Handling

3.5.1 Development and Runtime Error Reporting

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 `J1939RM_DEV_ERROR_DETECT == STD_ON`). Runtime errors are reported on the same way, if runtime error reporting is enabled (i.e. pre-compile parameter `J1939RM_RUNTIME_ERROR_REPORT == STD_ON`).

The reported J1939Rm module ID is 59 (0x3B).

The reported service IDs identify the services which are described in section 5.2 as well as the callback functions described in section 5.4. The following table presents the service IDs and the related services and callback functions:

Service ID		Service
0x01	J1939RM_SID_INIT	J1939Rm_Init()
0x02	J1939RM_SID_DEINIT	J1939Rm_DeInit()
0x03	J1939RM_SID_GETVERSIONINFO	J1939Rm_GetVersionInfo()
0x04	J1939RM_SID_MAINFUNCTION	J1939Rm_MainFunction()
0x05	J1939RM_SID_SETSTATE	J1939Rm_SetState()
0x07	J1939RM_SID_SENDREQUEST	J1939Rm_SendRequest()
0x08	J1939RM_SID_CANCELREQUESTTIMEOUT	J1939Rm_CancelRequestTimeout()
0x09	J1939RM_SID_SENDACK	J1939Rm_SendAck()
0x28	J1939RM_SID_COMRXIPDUCALLOUT	J1939Rm_ComRxIpduCallout()
0x40	J1939RM_SID_TXCONFIRMATION	J1939Rm_TxConfirmation()
0x42	J1939RM_SID_RXINDICATION	J1939Rm_RxIndication()
0x80	*J1939RM_SID_INITMEMORY*	J1939Rm_InitMemory()

Table 3-4 Service IDs

Note
The service IDs 0x80 and above are not specified by AUTOSAR.

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

Error Code		Description
0x01	J1939RM_E_UNINIT	An API was called while the module was uninitialized, i.e. before J1939Rm_Init or after J1939Rm_DeInit
0x02	J1939RM_E_REINIT	The Init API was called twice, i.e. after J1939Rm_Init and before J1939Rm_DeInit
0x03	J1939RM_E_PARAM_POINTER	An API service was called with a NULL pointer
0x04	J1939RM_E_INVALID_PDU_SDU_ID	An API service was called with a wrong ID
0x05	J1939RM_E_INVALID_NETWORK_ID	An API service was called with wrong network handle
0x06	J1939RM_E_INVALID_STATE	The API J1939Rm_SetState was called with a wrong state
0x07	J1939RM_E_INVALID_USER	An API was called with an illegal user ID
0x08	J1939RM_E_INVALID_PGN	An API was called with an unknown or illegal PGN
0x09	J1939RM_E_INVALID_PRIO	An API was called with an illegal priority
0x0a	J1939RM_E_INVALID_ADDRESS	An API was called with an illegal node address
0x0b	J1939RM_E_INVALID_OPTION	An API was called with an illegal Boolean option
0x0c	J1939RM_E_INVALID_ACK_CODE	An API was called with an illegal AckCode

Error Code		Description
0x0d	J1939RM_E_INVALID_NODE	An API was called with an illegal node ID
0x0e	J1939RM_E_INIT_FAILED	J1939Rm_Init called with invalid init structure
0x85	*J1939RM_E_INVALID_API*	An API was called that was not configured for the calling user
0x88	*J1939RM_E_INVALID_EXTID_INFO*	An API was called with invalid extended identifier bytes

Table 3-5 Development errors reported to DET

Note
The error codes 0x80 and above are not specified by AUTOSAR.

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

Error Code		Description
0x80	*J1939RM_E_INVALID_PDU_SIZE*	An illegal PDU size was reported by CanIf
0x81	*J1939RM_E_TIMEOUT_TXCONF*	Timeout of transmission confirmation callback
0x82	*J1939RM_E_ISOBUS_ACKM_ADDR*	DA of ISOBUS ACKM differs from AddressAcknowledged
0x83	*J1939RM_E_ACK_QUEUE_OVERRUN*	Acknowledgement message could not be stored in the queue
0x84	*J1939RM_E_REQ_QUEUE_OVERRUN*	Request message could not be stored in the queue
0x86	*J1939RM_E_INVALID_SPEC_INSTR*	Received Request2 message contains an invalid Special Instructions code
0x87	*J1939RM_E_INVALID_ACK_CB*	Received Acknowledgement message contains an invalid control byte
0x89	*J1939RM_E_BAD_PGN*	A PGN received within a Request, Request2, or Acknowledgment message has an invalid format
0x8a	*J1939RM_E_BAD_SA*	Received Request, Request2, or Acknowledgment message with an invalid source address
0x8b	*J1939RM_E_REQ2_QUEUE_OVERRUN*	Request2 message could not be stored in the queue

Table 3-6 Runtime errors reported to DET

Note
The error codes 0x80 and above are not specified by AUTOSAR.

4 Integration

This chapter gives necessary information for the integration of the MICROSAR J1939Rm into an application environment of an ECU.

4.1 Scope of Delivery

The delivery of the J1939Rm contains the files which are described in the sections 4.1.1 and 4.1.2:

4.1.1 Static Files

File Name	Description
J1939Rm.c	Implementation of the J1939Rm module
J1939Rm.h	Main header of the J1939Rm module
J1939Rm_Cbk.h	Callback header of the J1939Rm module
J1939Rm_Types.h	Global types header of the J1939Rm module
J1939Rm_Int.h	Internal header of the J1939Rm module

Table 4-1 Static files

4.1.2 Dynamic Files

The dynamic files are generated by DaVinci Configurator.

File Name	Description
J1939Rm_Cfg.h	Generated header file of J1939Rm containing pre-compile switches and providing symbolic defines
J1939Rm_Cfg.c	Generated source file of J1939Rm containing pre-compile time configurable parameters
J1939Rm_Lcfg.h	Generated header file of J1939Rm containing link time configurable preprocessor symbols
J1939Rm_Lcfg.c	Generated source file of J1939Rm containing link time configurable Parameters
J1939Rm_PBcfg.h	Generated header file of J1939Rm containing post-build time configurable preprocessor symbols
J1939Rm_PBcfg.c	Generated source file of J1939Rm containing post-build time configurable parameters

Table 4-2 Generated files

4.2 Critical Sections

The J1939Rm module uses critical sections to protect the state machines for the Tx PDUs and the Acknowledgement, Request and Request2 transmit queues:

> J1939RM_EXCLUSIVE_AREA_TXPDULOCK

> J1939RM_EXCLUSIVE_AREA_ACKQUEUELOCK

> J1939RM_EXCLUSIVE_AREA_REQ2QUEUELOCK

> J1939RM_EXCLUSIVE_AREA_REQ2QUEUELOCK

All these critical sections have a very short locking time, and do not cover any function calls.

Caution
To ensure data consistency, the exclusive areas of J1939Rm must be implemented as interrupt locks, and optimization must be disabled so that they are not removed by the RTE.

5 API Description

5.1 Type Definitions

The types defined by the J1939Rm are described in this section.

Type Name	C-Type	Description	Value Range
J1939Rm_AckCode	enum	This type represents the available kinds of acknowledgements.	`J1939RM_ACK_POSITIVE` Positive Acknowledgement (0)
			`J1939RM_ACK_NEGATIVE` Negative Acknowledgement (1)
			`J1939RM_ACK_ACCESS_DENIED` Access Denied (2)
			`J1939RM_ACK_CANNOT_RESPOND` Cannot Respond (3)
J1939Rm_ExtIdInfoType	struct	This type represents a set of extended identifiers.	`J1939Rm_ExtIdType ExtIdType` Extended identifier type
			`uint8 ExtId1` First extended identifier byte
			`uint8 ExtId2` Second extended identifier byte
			`uint8 ExtId3` Third extended identifier byte
J1939Rm_ExtIdType	enum	This type represents the available kinds of extended identifier usage.	`J1939RM_EXTID_NONE` No extended identifier bytes (0)
			`J1939RM_EXTID_ONE` One extended identifier byte (1)
			`J1939RM_EXTID_TWO` Two extended identifier bytes (2)
			`J1939RM_EXTID_THREE` Three extended identifier bytes (3)
			`J1939RM_EXTID_GF` Group function value in ACKM (4)
J1939Rm_StateType	enum	This type represents the communication state of the J1939 Request Manager.	`J1939RM_STATE_ONLINE` Normal communication (0)
			`J1939RM_STATE_OFFLINE` Only request for AC (1)

Table 5-1 Type definitions

5.2 Services provided by J1939Rm

This section describes the service functions that are implemented by the J1939Rm and can be invoked by other modules. The prototypes of the service functions are provided in the header file J1939Rm.h.

5.2.1 J1939Rm_InitMemory

Prototype	
`void J1939Rm_InitMemory (void)`	
Parameter	
none	
Return code	
void	
Functional Description	
Sets the global J1939Rm state to uninitialized.	
Particularities and Limitations	
This function should be used if the J1939Rm is not initialized by startup code.	
Call context	
Only to be called from initialization code.	

Table 5-2 J1939Rm_InitMemory

5.2.2 J1939Rm_Init

Prototype	
`void J1939Rm_Init (const J1939Rm_ConfigType *config)`	
Parameter	
config	Pointer to configuration data structure.
Return code	
void	
Functional Description	
Initializes the J1939 Request Manager.	
Particularities and Limitations	
The config parameter is only required if the configuration is variant or changed at post-build time.	
Call context	
Only to be called from task level.	
Preconditions	
The module must be in the uninitialized state.	

Table 5-3 J1939Rm_Init

5.2.3 J1939Rm_DeInit

Prototype	
`void J1939Rm_DeInit (void)`	
Parameter	
none	
Return code	
void	
Functional Description	
Resets the J1939 Request Manager to the uninitialized state.	
Particularities and Limitations	
The module is not truly shut down before all services and callback functions have terminated.	
Call context	
Only to be called from task level.	
Preconditions	
The module must be in the initialized state.	

Table 5-4 J1939Rm_DeInit

5.2.4 J1939Rm_GetVersionInfo

Prototype	
`void J1939Rm_GetVersionInfo (Std_VersionInfoType *versionInfo)`	
Parameter	
VersionInfo	Pointer to the location where the version information shall be stored.
Return code	
void	
Functional Description	
Returns the version information of the J1939 Request Manager.	
Particularities and Limitations	
none	
Call context	
May be called from interrupt or task level.	
Preconditions	
The VersionInfo parameter must not be NULL.	

Table 5-5 J1939Rm_GetVersionInfo

5.2.5 J1939Rm_SendRequest

Prototype
`Std_ReturnType J1939Rm_SendRequest (uint8 userId, NetworkHandleType channel,`

uint32 requestedPgn, J1939Rm_ExtIdInfoType *extIdInfo, uint8 destAddress, uint8 priority, boolean checkTimeout)	
Parameter	
userId	Identification of the calling module.
channel	Channel on which the request shall be sent.
requestedPgn	PGN of the requested PG.
extIdInfo	Extended identifier bytes.
destAddress	Address of the destination node or 0xFF for broadcast.
priority	Priority of the Request PG.
checkTimeout	TRUE: Timeout supervision will be performed, FALSE: No timeout supervision will be started
Return code	
Std_ReturnType	E_OK: Transmission request is accepted, E_NOT_OK: Transmission request is not accepted
Functional Description	
Requests transmission of a Request PG.	
Particularities and Limitations	
none	
Call context	
May be called from interrupt or task level.	

Table 5-6 J1939Rm_SendRequest

5.2.6 J1939Rm_CancelRequestTimeout

Prototype	
Std_ReturnType **J1939Rm_CancelRequestTimeout** (uint8 userId, NetworkHandleType channel, uint32 requestedPgn, J1939Rm_ExtIdInfoType *extIdInfo, uint8 destAddress)	
Parameter	
userId	Identification of the calling module.
channel	Channel on which the request shall be sent.
requestedPgn	PGN of the requested PG.
extIdInfo	Extended identifier bytes.
destAddress	Address of the destination node or 0xFF for broadcast.
Return code	
Std_ReturnType	E_OK: Cancellation of request timeout was successful, E_NOT_OK: Cancellation of request timeout was not successful
Functional Description	
Cancels timeout monitoring of a Request. If the request is not active, or timeout monitoring was not requested, this call has no effect.	

Particularities and Limitations
none
Call context
May be called from interrupt or task level.
Preconditions
The request identified by requested PGN and DA has been sent less than 1.25s in advance.

Table 5-7 J1939Rm_CancelRequestTimeout

5.2.7 J1939Rm_SendAck

Prototype	
`Std_ReturnType J1939Rm_SendAck (uint8 userId, NetworkHandleType channel, uint32 ackPgn, J1939Rm_ExtIdInfoType *extIdInfo, J1939Rm_AckCode ackCode, uint8 ackAddress, uint8 priority, boolean broadcast)`	
Parameter	
userId	Identification of the calling module.
channel	Channel on which the acknowledgement shall be sent.
ackPgn	Acknowledged PGN.
extIdInfo	Extended identifier bytes.
ackCode	Type of acknowledgement, see definition of J1939Rm_AckCode for available codes.
ackAddress	Address of the node that sent the request.
priority	Priority of the Acknowledgement PG.
broadcast	Indicates whether the ACKM is a response to a broadcast request.
Return code	
Std_ReturnType	E_OK: Transmission request is accepted, E_NOT_OK: Transmission request is not accepted
Functional Description	
Requests transmission of an Acknowledgement PG.	
Particularities and Limitations	
none	
Call context	
May be called from interrupt or task level.	

Table 5-8 J1939Rm_SendAck

5.2.8 J1939Rm_SetState

Prototype
`Std_ReturnType J1939Rm_SetState (NetworkHandleType channel, uint8 node, J1939Rm_StateType newState)`

Parameter	
channel	Channel for which the state shall be changed.
node	Node for which the state shall be changed.
newState	New state the J1939Rm shall enter, see definition of J1939Rm_StateType for available states.
Return code	
Std_ReturnType	E_OK: New communication state was set E_NOT_OK: Communication state was not changed due to wrong value in NewState or wrong initialization state of the module
Functional Description	
Changes the communication state of J1939Rm to offline (only Request for AC supported) or online.	
Particularities and Limitations	
none	
Call context	
May be called from interrupt or task level.	

Table 5-9 J1939Rm_SetState

5.3 Services used by J1939Rm

In the following table services provided by other components, which are used by the J1939Rm are listed. For details about prototype and functionality refer to the documentation of the providing component.

Component	API
COM	Com_TriggerIPDUSendWithMetaData
Default Error Tracer	Det_ReportError
PDU Router	PduR_J1939RmTransmit

Table 5-10 Services used by the J1939Rm

5.4 Callback Functions

This section describes the callback functions that are implemented by the J1939Rm and can be invoked by other modules. The prototypes of the callback functions are provided in the header file `J1939Rm_Cbk.h` by the J1939Rm.

5.4.1 J1939Rm_RxIndication

Prototype	
`void J1939Rm_RxIndication (PduIdType RxPduId, const PduInfoType *PduInfoPtr)`	
Parameter	
RxPduId	ID of the received I-PDU.
PduInfoPtr	Contains the length (SduLength) of the received I-PDU and a pointer to a buffer (SduDataPtr) containing the I-PDU and MetaData.

Return code	
void	
Functional Description	
Indicates the reception of a J1939Rm PDU from the PduR.	
Particularities and Limitations	
none	
Call context	
May be called from interrupt or task level.	
Preconditions	
J1939Rm_RxIndication is not currently executed with the same RxPduId.	

Table 5-11 J1939Rm_RxIndication

5.4.2 J1939Rm_TxConfirmation

Prototype	
`void J1939Rm_TxConfirmation (PduIdType TxPduId)`	
Parameter	
TxPduId	ID of the I-PDU that has been transmitted.
Return code	
void	
Functional Description	
Confirms the successful transmission of a J1939Rm PDU by the PduR.	
Particularities and Limitations	
none	
Call context	
May be called from interrupt or task level.	
Preconditions	
> J1939Rm_TxConfirmation is not currently executed with the same TxPduId. > Tx Confirmation Timeout did not expire for the same TxPduId.	

Table 5-12 J1939Rm_TxConfirmation

Caution

If Tx Confirmation Timeout expires for a certain TxPduId before the TxConfirmation is called for the same TxPduId, the behavior of J1939Rm cannot be predicted. Possible effects are the loss of queued Request, Request2 or Acknowledgement messages.

It is important to set the Tx Confirmation Timeout to a value larger than the longest transmission delay that can be expected on any referenced CAN bus.

5.4.3 J1939Rm_ComRxIpduCallout

Prototype	
`boolean J1939Rm_ComRxIpduCallout (PduIdType PduId, PduInfoType *PduInfoPtr)`	
Parameter	
PduId	ID of the received I-PDU.
PduInfoPtr	Contains the length (SduLength) of the received I-PDU and a pointer to the data of the I-PDU and MetaData (SduDataPtr).
Return code	
boolean	TRUE: I-PDU will be processed normal, FALSE: I-PDU will not be processed any further
Functional Description	
Indicates reception of a requested PDU by COM.	
Particularities and Limitations	
none	
Call context	
May be called from interrupt or task level.	
Preconditions	
J1939Rm_ComRxIpduCallout is not currently executed with the same PduId.	

Table 5-13 J1939Rm_ComRxIpduCallout

5.5 Configurable Interfaces

5.5.1 Notifications

At its configurable interfaces the J1939Rm defines notifications that can be mapped to callback functions provided by other modules. The mapping is not statically defined by the J1939Rm but can be performed at configuration time. The function prototypes that can be used for the configuration have to match the appropriate function prototype signatures, which are described in the following subsections.

5.5.1.1 <User>_RequestIndication

Prototype	
`void <User>_RequestIndication (uint8 node, NetworkHandleType channel, uint32 requestedPgn, J1939Rm_ExtIdInfoType *extIdInfo, uint8 sourceAddress, uint8 destAddress, uint8 priority)`	
Parameter	
node	Node by which the request was received.
channel	Channel on which the request was received.
requestedPgn	PGN of the requested PG.
extIdInfo	Extended identifier bytes.
sourceAddress	Address of the node that sent the Request PG.
destAddress	Address of this node or 0xFF for broadcast.

priority	Priority of the Request PG.
Return code	
`void`	
Functional Description	
Indicates reception of a Request PG.	
Particularities and Limitations	
none	
Call context	
May be called from interrupt or task level.	

Table 5-14 <User>_RequestIndication

5.5.1.2 <User>_AckIndication

Prototype	
`void <User>_AckIndication (uint8 node, NetworkHandleType channel, uint32 ackPgn, J1939Rm_AckCode ackCode, J1939Rm_ExtIdInfoType *extIdInfo, uint8 ackAddress, uint8 sourceAddress, uint8 priority)`	
Parameter	
node	Node by which the acknowledgement was received.
channel	Channel on which the acknowledgement was received.
ackPgn	Acknowledged PGN.
extIdInfo	Extended identifier bytes.
ackCode	Type of acknowledgement, see definition of J1939Rm_AckCode for available codes.
ackAddress	Address of this node.
sourceAddress	Address of the node that sent the Acknowledgement PG.
priority	Priority of the Acknowledgement PG.
Return code	
`void`	
Functional Description	
Indicates reception of an Acknowledgement PG.	
Particularities and Limitations	
none	
Call context	
May be called from interrupt or task level.	

Table 5-15 <User>_AckIndication

5.5.1.3 <User>_RequestTimeoutIndication

Prototype	
`void <User>_RequestTimeoutIndication (uint8 node, NetworkHandleType channel, uint32 requestedPgn, J1939Rm_ExtIdInfoType *extIdInfo, uint8 destAddress)`	
Parameter	
node	Node by which the request was sent.
channel	Channel on which the request was sent.
requestedPgn	PGN of the requested PG.
extIdInfo	Extended identifier bytes.
destAddress	Address of this node or 0xFF for broadcast.
Return code	
`void`	
Functional Description	
Indicates reception of a Request PG.	
Particularities and Limitations	
none	
Call context	
May be called from interrupt or task level.	

Table 5-16 <User>_RequestTimeoutIndication

5.6 Service Ports

J1939Rm has only client server interfaces. A client server interface is related to a Provide Port at the server side and a Require Port at client side.

5.6.1 Provide Ports on J1939Rm Side

At the Provide Ports of the J1939Rm, some of the API functions described in section 5.2 are available as Runnable Entities. The Runnable Entities are invoked via Operations. The mapping from an SWC client call to an Operation is performed by the RTE. With this mapping, the RTE adds Port Defined Argument Values to the client call of the SWC.

The following subsections present the Provide Ports defined for the J1939Rm and the Operations defined for the Provide Ports, the API functions related to the Operations and the Port Defined Argument Values to be added by the RTE.

5.6.1.1 J1939Rm_SendAck

Operation	API Function	Port Defined Argument Values
SendAck	J1939Rm_SendAck	uint8 userId

Table 5-17 J1939Rm_SendAck_{user}

5.6.1.2 J1939Rm_SendRequest

Operation	API Function	Port Defined Argument Values
SendRequest	J1939Rm_SendRequest	uint8 userId

Table 5-18 J1939Rm_SendRequest_{user}

5.6.1.3 J1939Rm_CancelRequestTimeout

Operation	API Function	Port Defined Argument Values
CancelRequestTimeout	J1939Rm_CancelRequestTimeout	uint8 userId

Table 5-19 J1939Rm_CancelRequestTimeout_{user}

5.6.2 Require Ports on J1939Rm Side

At its Require Ports the J1939Rm calls Operations. These Operations have to be provided by the SWCs by means of Runnable Entities. These Runnable Entities implement the callback functions expected by the J1939Rm.

The following subsections present the Require Ports defined for the J1939Rm, the Operations that are called from the J1939Rm and the related Notifications, which are described in section 5.5.1.

5.6.2.1 J1939Rm_AckIndication

Operation	Notification
AppAckIndication	<user>_AckIndication

Table 5-20 J1939Rm_AckIndication_{user}

5.6.2.2 J1939Rm_RequestIndication

Operation	Notification
AppRequestIndication	<user>_RequestIndication

Table 5-21 J1939Rm_RequestIndication_{user}

5.6.2.3 J1939Rm_RequestTimeoutIndication

Operation	Notification
RequestTimeoutIndication	<user>_ RequestTimeoutIndication

Table 5-22 J1939Rm_RequestTimeoutIndication_{user}

6 Configuration

6.1 Configuration Variants

The J1939Rm supports the configuration variants

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

The configuration classes of the J1939Rm parameters depend on the supported configuration variants. For their definitions please see the J1939Rm_bswmd.arxml file.

6.2 Post-Build Configuration

The configuration of post-build loadable is described in [5].

7 Glossary and Abbreviations

7.1 Glossary

Term	Description
DaVinci Configurator	Generation tool for MICROSAR components.

Table 7-1 Glossary

7.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
DET	Default Error Tracer
DP	Data Page, the most significant bit (MSB) of the 18 bit PGN
ECU	Electronic Control Unit
EDP	Extended Data Page, the second bit (after MSB) of the 18 bit PGN
HIS	Hersteller Initiative Software
MICROSAR	Microcontroller Open System Architecture (the Vector AUTOSAR solution)
PDUF	PDU Format, the middle byte of the 18 bit PGN
PDUS	PDU Specific, the lower byte of the 18 bit PGN
PGN	Parameter Group Number (18 bits, contains EDP, DP, PDUF, PDUS)
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