VECTOR

MICROSAR SAE J1939 Network Management

Technical Reference

Version 2.0.0

Authors	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	2014-06-19	0.2.0	Updated StartOfReception
Thomas Albrecht	2015-02-05	0.3.0	Added description of NAME API
Martin Schlodder	2015-03-23	0.3.1	Fixed description of BusOff callouts
Martin Schlodder	2015-09-10	1.0.0	First released version
Martin Schlodder	2016-01-12	1.0.1	Clarification on NvM handling
Martin Schlodder	2016-01-15	1.0.2	Improved description of NvM handling
Martin Schlodder	2016-03-14	1.0.3	Adapted J1939Nm_RequestIndication, added SIDs and DETs for NvM integration
Martin Schlodder	2016-05-09	1.0.4	Variant handling supported
Martin Schlodder	2017-05-02	2.0.0	Description of commanded NAME support

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	AUTOSAR_SWS_SAEJ1939NetworkManagement.pdf	4.2.1
[2]	AUTOSAR	AUTOSAR_SWS_DefaultErrorTracer.pdf	4.2.1
[3]	AUTOSAR	AUTOSAR_SWS_DiagnosticEventManager.pdf	4.2.1
[4]	AUTOSAR	AUTOSAR_SWS_BSWGeneral.pdf	4.2.1
[5]	AUTOSAR	AUTOSAR_TR_BSWModuleList.pdf	4.2.1
[6]	Vector	TechnicalReference_PostBuildLoadable.pdf	1.0.0
[7]	SAE J1939	J1939-81_2011-06.pdf	JUN2011

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]	Completed dynamic address claiming support
[0.3.0]	Added support for post-build configuration
[0.4.0]	Added NAME management API
[0.5.0]	Avoid BusOff by using random delays
[1.0.0]	Separate external and internal nodes, NvM support; component released
[2.0.0]	Adapted to changed signature of RequestIndication and SendRequest
[2.1.0]	Added support for variant handling
[3.0.0]	Commanded NAME supported via NameManagement message

Table 1-1 Component History

2 Introduction

This document describes the functionality, API and configuration of the AUTOSAR BSW module J1939Nm as specified in [1], with additional support for fully dynamic address arbitration.

Supported AUTOSAR Release:	4	
Supported Configuration Variants:	pre-compile, post-build-loadable, post-build-selectable	
Vendor ID:	J1939NM_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	J1939NM_MODULE_ID	34 decimal (according to [5])

The MICROSAR SAE J1939 Network Management implements the address arbitration defined in [7], including fully dynamic address handling, where addresses may change during runtime. It also supports the address allocation strategies defined in ISO 11783-5.

In the AUTOSAR architecture, the SAE J1939 Network Management module interacts with the CanIf to transmit and receive the AddressClaimed message, with J1939Rm to respond to requests for AddressClaimed, with the Nm interface for the state handling per channel, and with BswM to report state changes per node.

In case of fully dynamic address arbitration, the J1939Nm manages all addresses and NAMEs that are relevant for the communication of the node: The addresses and NAMEs of internal nodes, because they may change at runtime. And the addresses and NAMES of external nodes in order to track the communication partners.

To support fully dynamic address arbitration, J1939Nm also interacts with the PduR to receive the CommandedAddress message, with NvM to store the current NAME and address of internal nodes persistently, and with CanIf to maintain address translation tables and to support NAME changes using the NameManagement message. CDDs and integration code can also access the NAMEs and addresses of all internal and external nodes, search them, and change those of internal nodes.

J1939Nm uses the Meta Data support introduced with AUTOSAR 4.1.1 to reduce the number of AddressClaimed and CommandedAddress PDUs to at most one per channel and direction.

2.1 Architecture Overview

The following figure highlights the position of the J1939Nm in the AUTOSAR architecture.

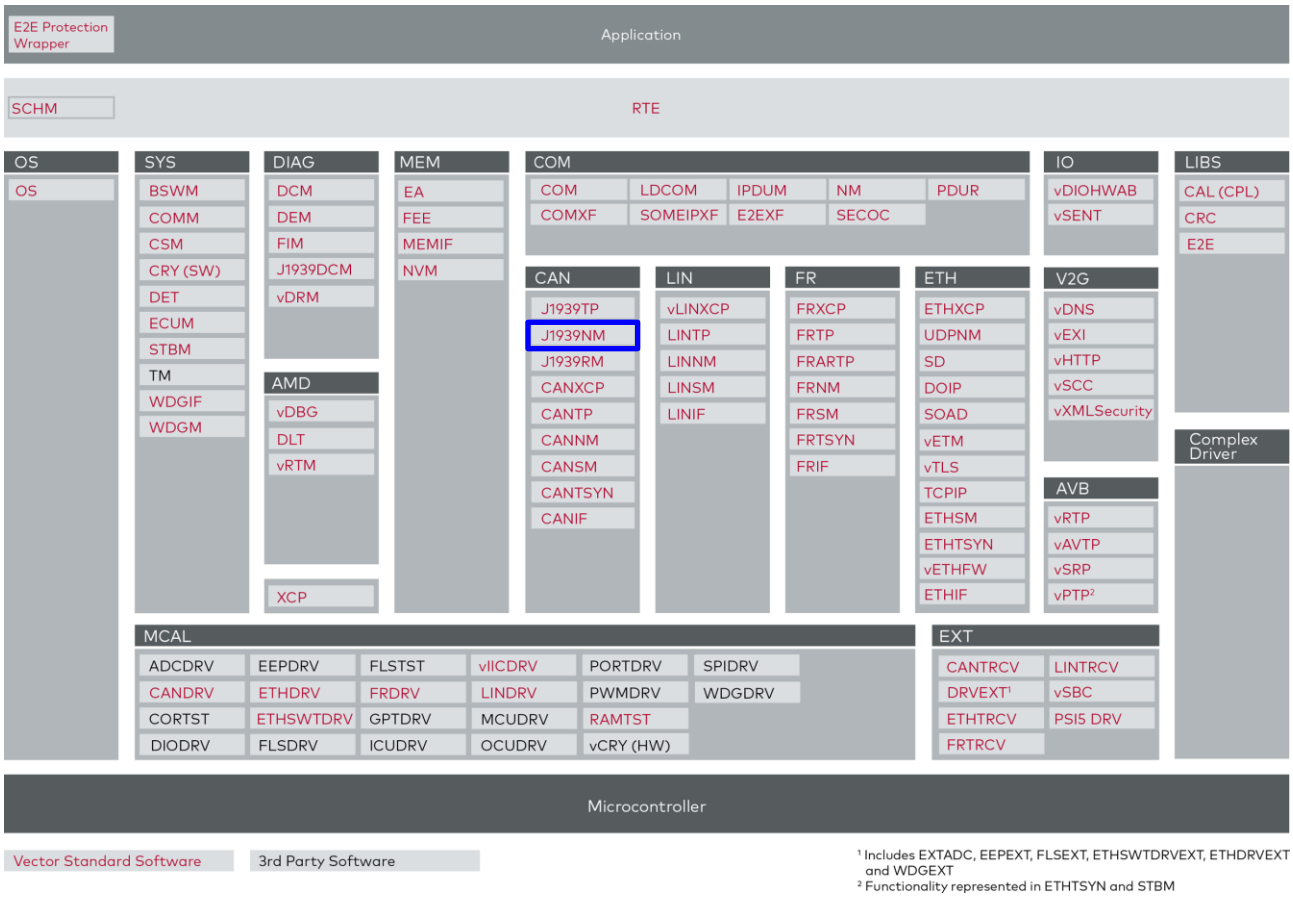

Figure 2-1 AUTOSAR 4.2 Architecture Overview

3 Functional Description

3.1 Features

The features listed in the following tables cover the complete functionality specified for the J1939Nm.

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

> Table 3-1 Supported AUTOSAR standard conform features

Vector Informatik provides further J1939Nm functionality beyond the AUTOSAR standard. The corresponding features are listed in the table

> Table 3-2 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
Network management state handling
Transmission of AddressClaimed messages
Reception of AddressClaimed messages
Request for AddressClaimed
Production error reporting
Meta data handling

Table 3-1 Supported AUTOSAR standard conform features

3.1.1 Additions/ Extensions

The following features are provided beyond the AUTOSAR standard:

Features Provided Beyond The AUTOSAR Standard
Extended error reporting
Fully dynamic address allocation and address tracking
API for access to NAMEs and addresses of internal and external nodes
NAME changes with the NameManagement message

Table 3-2 Features provided beyond the AUTOSAR standard

3.1.1.1 Extended Error Reporting

The J1939Nm reports additional development errors that are not specified by AUTOSAR. See Table 3-4 in section 3.5.1.

3.1.1.2 Fully Dynamic Address Allocation and Address Tracking

The J1939Nm can be configured to support fully dynamic address arbitration. In this case, the addresses of all messages except for the AddressClaimed message are translated by

CanIf on-the-fly from an internal representation to the actually used address and vice versa. The J1939Nm keeps track of the current NAME and address of all internal and external nodes, and preserves the current NAME and address of internal nodes using NvM.

3.1.1.3 API for Access to NAMEs and Addresses of Internal and External Nodes

J1939Nm provides a dedicated API to change the NAME of internal nodes in a static network, and to get information about currently available nodes (internal and external) and to change the address of internal nodes in a dynamic network.

3.1.1.4 NAME changes with the NameManagement message

J1939Nm allows changing the NAME of an internal node from an external node by use of the NameManagement message. The NameManagement message can also be used to trigger the AddressClaimed message based on certain NAME fields, or to acquire the current or pending NAME of a node identified by its address.

Note
A pending NAME times out 1.25s after the last command from the tester.

3.1.2 Limitations

3.1.2.1 Concurrent CommandedAddress Handling

J1939Nm can only handle one CommandedAddress message at a time. This is partly caused by just one PDU per channel, which means that J1939Tp cannot receive more than one CommandedAddress from one channel at the same time. And partly due to the state machine, which is currently implemented centrally and only once, so that a second CommandedAddress from a second channel at the same time will be rejected.

3.1.2.2 Concurrent NAME Changes

J1939Nm can only handle one NAME change at a time, either triggered by the NAME API, or via the NameManagement message. The reason is that there is only one state machine, which is implemented centrally for all internal nodes. A NAME change using J1939Nm_SetName() will be rejected when a NAME change via NameManagement messages has not been finished, and NAME changes via NameManagement messages will be rejected when another NAME change for another node or another channel is currently going on, or when J1939Nm_SetName() is currently being called.

3.2 Initialization

The J1939Nm uses a global state (J1939Nm_ModuleInitialized) to determine whether the module is initialized and operational. This state is initially set to J1939NM_UNINIT. If initialization by startup code is not supported, the initialization routine should call J1939Nm_InitMemory() to set the global state to J1939NM_UNINIT.

If dynamic address arbitration is enabled, NvM has to be initialized before J1939Nm, and NvM_ReadAll() must have been called.

Caution
If dynamic address arbitration is enabled, J1939Nm can only be initialized after the NM-RAM read-back initiated by NvM_ReadAll() has completed, successfully or not. Otherwise, the initialization of J1939Nm will fail.

By calling J1939Nm_Init(), the J1939Nm module is set to the state J1939NM_INIT, and internal states are set to their initial states. The module is now operational.

To stop the J1939Nm module, J1939Nm_DeInit() may be called, which sets the global state to J1939NM_UNINIT again.

3.3 States

The J1939Nm module has a global state and separate states for each Tx PDU and for each internal node. If dynamic address arbitration is enabled, J1939Nm also maintains separate states for each CommandedAddress PDU, each channel, and each external node.

3.3.1 Global State

The global state is switched by the services J1939Nm_InitMemory(), J1939Nm_Init(), and J1939Nm_DeInit().

In the state J1939NM_UNINIT, all services of J1939Nm return immediately. If they have a return value, an error (typically E_NOT_OK) is returned.

In the state J1939NM_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 J1939Nm_TxConfirmation() being called after each call to CanIf_Transmit(). Because this is not always the case, the J1939Nm monitors this state and resets it after a timeout configurable via “Tx Confirmation Timeout”, assuming the PDU was not transmitted.

3.3.3 CommandedAddress PDU State

J1939Nm maintains a global state machine for reception of CommandedAddress PDUs.

3.3.4 NameManagement State

J1939Nm maintains a global state machine for handling NAME changes using the NameManagement message.

3.3.5 Channel State

J1939Nm uses separate state machines per channel to synchronize address refresh cycles and to monitor allocated addresses.

3.3.6 Internal Node State

J1939Nm maintains a separate state for each internal node to track the NM state and mode and an internal sub state. These are used to handle the address claiming procedure.

3.3.7 External Node State

Each external node has a state machine which tracks the current availability of the node.

3.4 Main Function

The J1939Nm_MainFunction() is used by the J1939Nm module to supervise all kinds of timing. Therefore, it is essential that this main function is called with the timing configured via "Main Function Period".

If the timing is not exact, a typical error will be that the ECU goes online too early or too late after an address claim.

3.5 Error Handling

3.5.1 Development 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 `J1939NM_DEV_ERROR_DETECT == STD_ON`).

The reported J1939Nm module ID is 34 (0x22).

The reported service IDs identify the services which are described in section 5.1 as well as the callback functions described in section 5.3. The following table presents the service IDs and the related services and callback functions:

Service ID		Service
0x01	J1939NM_SID_INIT	J1939Nm_Init()
0x02	J1939NM_SID_DEINIT	J1939Nm_DeInit()
0x03	J1939NM_SID_GETVERSIONINFO	J1939Nm_GetVersionInfo()
0x04	J1939NM_SID_MAINFUNCTION	J1939Nm_MainFunction()
0x05	J1939NM_SID_NETWORKREQUEST	J1939Nm_NetworkRequest()
0x06	J1939NM_SID_NETWORKRELEASE	J1939Nm_NetworkRelease()
0x0d	J1939NM_SID_GETSTATE	J1939Nm_GetState()
0x0f	J1939NM_SID_PASSIVESTARTUP	J1939Nm_PassiveStartup()
0x10	J1939NM_SID_GETBUSOFFDELAY	J1939Nm_GetBusOffDelay()
0x40	J1939NM_SID_TXCONFIRMATION	J1939Nm_TxConfirmation()
0x42	J1939NM_SID_RXINDICATION	J1939Nm_RxIndication()
0x43	J1939NM_SID_REQUESTINDICATION	J1939Nm_RequestIndication()
0x80	J1939NM_SID_INITMEMORY	J1939Nm_InitMemory()
0x81	J1939NM_SID_STARTOFRECEPTION	J1939Nm_StartOfReception()
0x82	J1939NM_SID_COPYRXDATA	J1939Nm_CopyRxData()
0x83	J1939NM_SID_TPRXINDICATION	J1939Nm_TpRxIndication()
0x85	J1939NM_SID_BUSOFFEND	J1939Nm_BusOffEnd()
0x86	J1939NM_SID_GETNODE	J1939Nm_GetNode()
0x87	J1939NM_SID_GETFIRSTUNKNOWNNAMEIDX	J1939Nm_GetFirstUnknownNameIdx()
0x88	J1939NM_SID_GETLASTNODEIDX	J1939Nm_GetLastNodeIdx()

Service ID		Service
0x89	J1939NM_SID_FINDNODEBYNAME	J1939Nm_FindNodeByName()
0x8a	J1939NM_SID_FINDNODEBYADDRESS	J1939Nm_FindNodeByAddress()
0x8b	J1939NM_SID_SETNAME	J1939Nm_SetName()
0x8c	J1939NM_SID_SETADDRESS	J1939Nm_SetAddress()
0x8d	J1939NM_SID_NVMINIT_CURRENTNODE-ADDRESSES	J1939Nm_NvMInit_CurrentNode-Addresses()
0x8e	J1939NM_SID_NVMINIT_CURRENTNODENAMES	J1939Nm_NvMInit_CurrentNodeNames()

Table 3-3 Service IDs

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

Error Code		Description
0x01	J1939NM_E_UNINIT	An API was called while the module was uninitialized, i.e. before J1939Nm_Init or after J1939Nm_DeInit
0x02	J1939NM_E_REINIT	The Init API was called twice, i.e. after J1939Nm_Init and before J1939Nm_DeInit
0x03	J1939NM_E_PARAM_POINTER	An API service was called with a NULL pointer
0x04	J1939NM_E_INVALID_PDU_SDU_ID	An API service was called with a wrong ID
0x05	J1939NM_E_INVALID_NETWORK_ID	An API service was called with wrong network handle
0x06	J1939NM_E_INVALID_PGN	An API was called with an unsupported PGN
0x07	J1939NM_E_INVALID_PRIO	An API was called with an illegal priority
0x08	J1939NM_E_INVALID_ADDRESS	An API was called with an illegal node address
0x09	J1939NM_E_INVALID_NODE	An API was called with an illegal node ID
0x0a	J1939NM_E_INIT_FAILED	J1939Nm_Init called with invalid init structure
0x80	*J1939NM_E_INVALID_PDU_SIZE*	An illegal PDU size was reported by CanIf
0x81	*J1939NM_E_TIMEOUT_TXCONF*	Timeout of transmission confirmation callback
0x82	*J1939NM_E_DUPLICATE_NAME*	A remote node uses the same NAME as a local node
0x83	*J1939NM_E_EXTERNAL_NODE*	API not supported for remote nodes
0x84	*J1939NM_E_RUNNING*	An NvM initialization API was called in initialized state
0x90	*J1939NM_E_DUMMY_API*	A dummy API was called

Table 3-4 Errors reported to DET

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

3.5.2 Production Code Error Reporting

Production errors are reported to the DEM using the service `Dem_ReportErrorStatus()` as specified in [3], if production error reporting is enabled (i.e. pre-compile parameter `J1939NM_PROD_ERROR_DETECT == STD_ON`).

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

Error Code	Description
J1939NM_E_ADDRESS_LOST	The desired address could not be claimed.

Table 3-5 Errors reported to DEM

4 Integration

This chapter gives necessary information for the integration of the MICROSAR J1939Nm into an application environment of an ECU.

4.1 Scope of Delivery

The delivery of the J1939Nm contains the files which are described in the sections 4.1.1 and 4.1.2:

4.1.1 Static Files

File Name	Description
J1939Nm.c	Implementation of the J1939Nm module
J1939Nm.h	Main header of the J1939Nm module
J1939Nm_Cbk.h	Callback header of the J1939Nm module
J1939Nm_NvM.h	NvM header of the J1939Nm module
J1939Nm_Types.h	Global types header of the J1939Nm module
J1939Nm_Int.h	Internal header of the J1939Nm module
J1939Nm_Dynamic.c	Implementation of the dynamic sub-module of the J1939Nm module
J1939Nm_Dynamic.h	Header of the dynamic sub-module of the J1939Nm module

Table 4-1 Static files

The files J1939Nm_NvM.h, J1939Nm_Dynamic.c, and J1939Nm_Dynamic.h are only included in the delivery if dynamic addressing is licensed.

4.1.2 Dynamic Files

The dynamic files are generated by DaVinci Configurator.

File Name	Description
J1939Nm_Cfg.h	Generated header file of J1939Nm containing pre-compile switches and providing symbolic defines
J1939Nm_Cfg.c	Generated source file of J1939Nm containing pre-compile time configurable parameters
J1939Nm_Lcfg.h	Generated header file of J1939Nm containing link time configurable preprocessor symbols
J1939Nm_Lcfg.c	Generated source file of J1939Nm containing link time configurable Parameters
J1939Nm_PBcfg.h	Generated header file of J1939Nm containing post-build time configurable preprocessor symbols
J1939Nm_PBcfg.c	Generated source file of J1939Nm containing post-build time configurable parameters

Table 4-2 Generated files

4.2 Critical Sections

The J1939Nm module uses critical sections to protect the state machines for the Tx PDUs, the CommandedAddress PDUs, and the current NAME table:

- J1939Nm_TxPduLock
- J1939Nm_RxPduLock
- J1939Nm_NameLock
- J1939Nm_NmActiveLock
- J1939Nm_NmQueueLock

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

5 API Description

5.1 Services provided by J1939Nm

This section describes the service functions that are implemented by the J1939Nm and can be invoked by other modules. The prototypes of the service functions are provided in the header file J1939Nm.h.

5.1.1 J1939Nm_InitMemory

Prototype	
`void J1939Nm_InitMemory (void)`	
Parameter	
none	
Return code	
void	
Functional Description	
Sets the global J1939Nm state to uninitialized.	
Particularities and Limitations	
This function should be used if the J1939Nm is not initialized by startup code.	
Call context	
Only to be called from initialization code.	

Table 5-1 J1939Nm_InitMemory

5.1.2 J1939Nm_Init

Prototype	
`void J1939Nm_Init (const J1939Nm_ConfigType *config)`	
Parameter	
config	Pointer to configuration data structure.
Return code	
void	
Functional Description	
Initializes the J1939 Network Management module.	
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.	
> NvM_ReadAll() must have been called.	

Table 5-2 J1939Nm_Init

5.1.3 J1939Nm_DeInit

Prototype	
`void J1939Nm_DeInit (void)`	
Parameter	
none	
Return code	
void	
Functional Description	
Resets the J1939 Network Management module 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 5-3 J1939Nm_DeInit

5.1.4 J1939Nm_GetVersionInfo

Prototype	
`void J1939Nm_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 this module.	
Particularities and Limitations	
none	
Call context	
May be called from interrupt or task level.	
Preconditions	
The VersionInfo parameter must not be NULL.	

Table 5-4 J1939Nm_GetVersionInfo

5.1.5 J1939Nm_NetworkRequest

Prototype	
`Std_ReturnType J1939Nm_NetworkRequest (const NetworkHandleType channel)`	
Parameter	
channel	Identification of the NM-channel.
Return code	
Std_ReturnType	E_OK: No error, E_NOT_OK: Requesting of network has failed.
Functional Description	
Request the network, since ECU needs to communicate on the bus.	
Particularities and Limitations	
none	
Call context	
May be called from interrupt or task level.	

Table 5-5 J1939Nm_NetworkRequest

5.1.6 J1939Nm_NetworkRelease

Prototype	
`Std_ReturnType J1939Nm_NetworkRelease (const NetworkHandleType channel)`	
Parameter	
channel	Identification of the NM-channel.
Return code	
Std_ReturnType	E_OK: No error, E_NOT_OK: Releasing of network has failed.
Functional Description	
Release the network, since ECU does not have to communicate on the bus.	
Particularities and Limitations	
none	
Call context	
May be called from interrupt or task level.	

Table 5-6 J1939Nm_NetworkRelease

5.1.7 J1939Nm_GetState

Prototype	
`Std_ReturnType J1939Nm_GetState (const NetworkHandleType channel, Nm_StateType *const stateP, Nm_ModeType *const modeP)`	
Parameter	
channel	Identification of the NM-channel.

stateP	Pointer where state of the network management shall be copied to.
modeP	Pointer where the mode of the network management shall be copied to.
Return code	
Std_ReturnType	E_OK: No error, E_NOT_OK: Getting of NM state has failed.
Functional Description	
Returns the state and the mode of the network management.	
Particularities and Limitations	
none	
Call context	
May be called from interrupt or task level.	
Preconditions	
The pointer parameters must not be NULL.	

Table 5-7 J1939Nm_GetState

5.1.8 J1939Nm_GetNode

Prototype	
`Std_ReturnType` **`J1939Nm_GetNode`** `(NetworkHandleType channel, uint16 nodeId, boolean external, uint8 *name, uint8 *virtAddr, uint8 *busAddr)`	
Parameter	
channel	Communication channel.
nodeId	Node ID.
external	TRUE: Look for an external node. FALSE: Look for an internal node.
name	Pointer to an array of 8 bytes for the 64-bit NAME. If a NULL pointer is provided, the NAME will not be copied.
virtAddr	Virtual address of the node, used in MetaData.
busAddr	Actual address of the node as seen on the bus.
Return code	
Std_ReturnType	E_OK: The node lookup was successful. E_NOT_OK: The node could not be located, e.g. because the provided nodeId or channel are wrong, the node is not of the required type (internal/external), the node is not attached to the channel, or the node is an external node and currently not online.
Functional Description	
Copies the NAME, the virtual address, and the bus address of the internal or external node designated by node ID and channel.	
Particularities and Limitations	
Part of NAME access feature. Only available if dynamic addressing is licensed.	
Call context	

May be called from interrupt or task level.
Preconditions
The pointer parameters apart from name must not be NULL.

Table 5-8 J1939Nm_GetNode

5.1.9 J1939Nm_GetFirstUnknownNameIdx

Prototype	
`uint16 J1939Nm_GetFirstUnknownNameIdx (NetworkHandleType channel)`	
Parameter	
channel	Communication channel.
Return code	
uint16	Index of first external node with an unknown NAME. If no such nodes are configured, 65535 will be returned.
Functional Description	
Returns the node ID of the first external node with an unknown NAME.	
Particularities and Limitations	
Part of NAME access feature. Only available if dynamic addressing is licensed.	
Call context	
May be called from interrupt or task level.	

Table 5-9 J1939Nm_GetFirstUnknownNameIdx

5.1.10 J1939Nm_GetLastNodeIdx

Prototype	
`uint16 J1939Nm_GetLastNodeIdx (NetworkHandleType channel)`	
Parameter	
channel	Communication channel.
Return code	
uint16	Index of the last external node available on the given channel.
Functional Description	
Returns the ID of the last currently available external node attached to the given channel.	
Particularities and Limitations	
Part of NAME access feature. Only available if dynamic addressing is licensed.	
Call context	
May be called from interrupt or task level.	

Table 5-10 J1939Nm_GetLastNodeIdx

5.1.11 J1939Nm_FindNodeByName

Prototype
`Std_ReturnType J1939Nm_FindNodeByName (NetworkHandleType channel, boolean external, const uint8 *name, const uint8 *mask, uint16 *nodeId)`
Parameter
channel — Communication channel.
external — TRUE: Look for an external node. FALSE: Look for an internal node.
name — Pointer to an array of 8 bytes containing the 64-bit NAME to be looked for.
mask — Pointer to an array of 8 bytes to define the relevant name parts for the search. Only those bits of the name are checked which are set to 1 in the mask.
nodeId — The first node ID to look for. On success, the value is changed to the found node ID.
Return code
Std_ReturnType — E_OK: The node lookup was successful. E_NOT_OK: No suitable node was found, or the provided channel does not exist.
Functional Description
Looks for an internal or external node attached to the provided channel that matches the provided name and mask. External nodes are only detected if they sent an AddressClaimed or CannotClaimAddress after the last request for AC. Providing a 0 for the node ID will find the first matching node. To implement an iterative search, the node ID provided to the next call should be set to the successor of the last found node.
Particularities and Limitations
Part of NAME access feature. Only available if dynamic addressing is licensed.
Call context
May be called from interrupt or task level.
Preconditions
The pointer parameters must not be NULL.

Table 5-11 J1939Nm_FindNodeByName

5.1.12 J1939Nm_FindNodeByAddress

Prototype
`Std_ReturnType J1939Nm_FindNodeByAddress (NetworkHandleType channel, boolean external, uint8 virtAddr, uint16 *nodeId)`
Parameter
channel — Communication channel.
external — TRUE: Look for an external node. FALSE: Look for an internal node.
virtAddr — Virtual address of the wanted node.
nodeId — Found node ID.

Return code	
Std_ReturnType	E_OK: The node lookup was successful. E_NOT_OK: No suitable node was found, or the provided channel does not exist.
Functional Description	
Looks for an internal or external node that matches the provided virtual address. External nodes are only detected if they sent an AddressClaimed or CannotClaimAddress after the last request for AC.	
Particularities and Limitations	
Part of NAME access feature. Only available if dynamic addressing is licensed.	
Call context	
May be called from interrupt or task level.	
Preconditions	
The pointer parameter must not be NULL.	

Table 5-12 J1939Nm_FindNodeByAddress

5.1.13 J1939Nm_SetName

Prototype	
`Std_ReturnType J1939Nm_SetName (NetworkHandleType channel, uint16 nodeId, boolean external, const uint8 *name)`	
Parameter	
channel	Communication channel.
nodeId	Node ID.
external	TRUE: Change NAME of an external node using NAME management messages. FALSE: Change NAME of an internal node.
name	Pointer to an array of 8 bytes containing the new 64-bit NAME.
Return code	
Std_ReturnType	E_OK: The NAME change was successful. E_NOT_OK: The NAME could not be changed, e.g. because the provided nodeId or channel were wrong, or the node is an external node.
Functional Description	
Sets a new NAME for the designated internal node. The node will claim the new NAME afterwards. External nodes are not yet supported.	
Particularities and Limitations	
Part of NAME access feature. Only available if dynamic addressing is licensed.	
Call context	
May be called from interrupt or task level.	
Preconditions	
The pointer parameter must not be NULL.	

Table 5-13 J1939Nm_SetName

5.1.14 J1939Nm_SetAddress

Prototype	
`Std_ReturnType J1939Nm_SetAddress (NetworkHandleType channel, uint16 nodeId, boolean external, uint8 busAddr)`	
Parameter	
channel	Communication channel.
nodeId	Node ID.
external	TRUE: Change address of an external node using CommandedAddress. FALSE: Change address of an internal node.
busAddr	The new actual address of the node.
Return code	
Std_ReturnType	E_OK: The address change was successful. E_NOT_OK: The address could not be changed, e.g. because the provided nodeId or channel were wrong, or the node is an external node.
Functional Description	
Sets a new address for the designated internal node. The node will claim the new address afterwards. External nodes are not yet supported.	
Particularities and Limitations	
Part of NAME access feature. Only available if dynamic addressing is licensed.	
Call context	
May be called from interrupt or task level.	

Table 5-14 J1939Nm_SetAddress

5.1.15 J1939Nm_PassiveStartUp

Prototype	
`Std_ReturnType J1939Nm_PassiveStartUp (const NetworkHandleType channel)`	
Parameter	
channel	Identification of the NM-channel.
Return code	
Std_ReturnType	E_OK: No error, E_NOT_OK: Passive startup of network management has failed.
Functional Description	
Passive startup of the NM. It triggers the transition from Bus-Sleep Mode to the Network Mode without requesting the network.	
Particularities and Limitations	
Dummy function.	
Call context	
May be called from interrupt or task level.	

Table 5-15 J1939Nm_PassiveStartUp

5.2 Services used by J1939Nm

In the following table services provided by other components, which are used by the J1939Nm are listed. For details about prototype and functionality refer to the documentation of the providing component.

Component	API
CAN Interface	CanIf_Transmit
	CanIf_ResetAddressTableEntry
	CanIf_SetAddressTableEntry
Default Error Tracer	Det_ReportError
Diagnostic Event Manager	Dem_ReportErrorStatus
NVRAM Manager	NvM_GetErrorStatus
	NvM_SetRamBlockStatus
SAE J1939 Request Manager	J1939Rm_SendRequest

Table 5-16 Services used by the J1939Nm

5.3 Callback Functions

This section describes the callback functions that are implemented by the J1939Nm and can be invoked by other modules. The prototypes of the following callback functions are provided in the header file `J1939Nm_Cbk.h` by the J1939Nm.

5.3.1 J1939Nm_RxIndication

Prototype	
`void J1939Nm_RxIndication (PduIdType id, const PduInfoType *info)`	
Parameter	
id	ID of the received NM-PDU.
info	Contains the length (SduLength) of the received NM-PDU and a pointer to a buffer (SduDataPtr) containing the NM-PDU and MetaData.
Return code	
void	
Functional Description	
Indicates the reception of an AddressClaimed NM-PDU from the CanIf.	
Particularities and Limitations	
none	
Call context	
May be called from interrupt or task level.	
Preconditions	
J1939Nm_RxIndication is not currently executed with the same id.	

Table 5-17 J1939Nm_RxIndication

5.3.2 J1939Nm_TxConfirmation

Prototype	
`void J1939Nm_TxConfirmation (PduIdType TxPduId)`	
Parameter	
TxPduId	ID of the NM-PDU that has been transmitted.
Return code	
void	
Functional Description	
Confirms the successful transmission of an AddressClaimed NM-PDU by the CanIf.	
Particularities and Limitations	
none	
Call context	
May be called from interrupt or task level.	
Preconditions	
J1939Nm_TxConfirmation is not currently executed with the same TxPduId.	

Table 5-18 J1939Nm_TxConfirmation

5.3.3 J1939Nm_RequestIndication

Prototype	
`void J1939Nm_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 5-19 J1939Nm_RequestIndication

5.3.4 J1939Nm_StartOfReception

Prototype	
`BufReq_ReturnType` **`J1939Nm_StartOfReception`** `(PduIdType id, const PduInfoType *info, PduLengthType TpSduLength, PduLengthType *bufferSizePtr)`	
Parameter	
id	Identification of the N-SDU.
info	Pointer to a PduInfoType structure containing the MetaData and MetaDataLength.
TpSduLength	Total length of the N-SDU to be received.
bufferSizePtr	Available receive buffer in the receiving module. This parameter will be used to compute the block size in the transport protocol module.
Return code	
BufReq_ReturnType	BUFREQ_OK: Connection has been accepted. bufferSizePtr indicates the available receive buffer; reception is continued. If no buffer of the requested size is available, a receive buffer size of 0 shall be indicated by bufferSizePtr. BUFREQ_E_NOT_OK: Connection has been rejected; reception is aborted. bufferSizePtr remains unchanged. BUFREQ_E_OVFL: No buffer of the required length can be provided; reception is aborted. bufferSizePtr remains unchanged.
Functional Description	
This function is called at the start of receiving a CommandedAddress N-SDU from J1939Tp.	
Particularities and Limitations	
Only available if dynamic addressing is licensed.	
Call context	
May be called from interrupt or task level.	
Preconditions	
No N-SDU reception is currently active with the same id.	

Table 5-20 J1939Nm_StartOfReception

5.3.5 J1939Nm_CopyRxData

Prototype	
`BufReq_ReturnType` **`J1939Nm_CopyRxData`** `(PduIdType id, PduInfoType *info, PduLengthType *availableDataPtr)`	
Parameter	
id	Identification of the received N-SDU.
info	Provides the source buffer (SduDataPtr) and the number of bytes to be copied (SduLength). An SduLength of 0 indicates a query for the current amount of available buffer in the upper layer module. In this case, the SduDataPtr may be a NULL_PTR.
bufferSizePtr	Available receive buffer after data has been copied.

Return code	
BufReq_ReturnType	BUFREQ_OK: Data copied successfully. BUFREQ_E_NOT_OK: Data was not copied because an error occurred.
Functional Description	
This function is called to provide the received data of an N-SDU segment (N-PDU) to the upper layer. Each call to this function provides the next part of the N-SDU data. The size of the remaining data is written to the position indicated by bufferSizePtr.	
Particularities and Limitations	
Only available if dynamic addressing is licensed.	
Call context	
May be called from interrupt or task level.	
Preconditions	
The reception of the N-SDU with this id was previously accepted using J1939Nm_StartOfReception.	

Table 5-21 J1939Nm_CopyRxData

5.3.6 J1939Nm_TpRxIndication

Prototype	
`void J1939Nm_TpRxIndication (PduIdType id, Std_ReturnType result)`	
Parameter	
id	Identification of the received N-SDU.
result	Result of the reception.
Return code	
void	
Functional Description	
Called after a CommandedAddress N-SDU has been received via the TP API, the result indicates whether the reception was successful or not.	
Particularities and Limitations	
Only available if dynamic addressing is licensed.	
Call context	
May be called from interrupt or task level.	
Preconditions	
The reception of the N-SDU with this id was previously accepted using J1939Nm_StartOfReception.	

Table 5-22 J1939Nm_TpRxIndication

5.3.7 J1939Nm_GetBusOffDelay

Prototype
`void J1939Nm_GetBusOffDelay (NetworkHandleType channel, uint8 *delayCyclesPtr)`

Parameter	
channel	ComM network ID of the affected channel.
delayCyclesPtr	Pointer to the location where the number of delay cycles shall be stored.
Return code	
void	
Functional Description	
Called when a bus-off was detected by the CanSM, returns the number of CanSM main cycles to delay the recovery.	
Particularities and Limitations	
none	
Call context	
May be called from interrupt or task level.	

Table 5-23 J1939Nm_GetBusOffDelay

5.3.8 J1939Nm_BusOffEnd

Prototype	
`void` **`J1939Nm_BusOffEnd`** `(NetworkHandleType channel)`	
Parameter	
channel	ComM network ID of the affected channel.
Return code	
void	
Functional Description	
Called by CanSM after bus-off recovery succeeded.	
Particularities and Limitations	
none	
Call context	
May be called from interrupt or task level.	

Table 5-24 J1939Nm_BusOffEnd

The prototypes of the following callback functions are provided in the header file `J1939Nm_NvM.h` by the J1939Nm.

5.3.9 J1939Nm_NvMInit_CurrentNodeAddresses

Prototype	
`Std_ReturnType` **`J1939Nm_NvMInit_CurrentNodeAddresses`** `(void)`	
Parameter	
none	

Return code	
Std_ReturnType	E_OK: The table was successfully reset. E_NOT_OK: The table reset failed.
Functional Description	
Resets the current address table to the preferred addresses of the nodes.	
Particularities and Limitations	
This function is called by NvM to initialize the table if ReadAll fails to restore saved values. Only available if dynamic addressing is licensed.	
Call context	
May be called from interrupt or task level.	
Preconditions	
The module must not be in the initialized state.	

Table 5-25 J1939Nm_NvMInit_CurrentNodeAddresses

5.3.10 J1939Nm_NvMInit_CurrentNodeNames

Prototype	
`Std_ReturnType` **`J1939Nm_NvMInit_CurrentNodeNames`** `(void)`	
Parameter	
none	
Return code	
Std_ReturnType	E_OK: The table was successfully reset. E_NOT_OK: The table reset failed.
Functional Description	
Resets the current NAME table to the configured NAMEs of the nodes.	
Particularities and Limitations	
This function is called by NvM to initialize the table if ReadAll fails to restore saved values. Only available if dynamic addressing is licensed.	
Call context	
May be called from interrupt or task level.	
Preconditions	
The module must not be in the initialized state.	

Table 5-26 J1939Nm_NvMInit_CurrentNodeNames

5.4 Configurable Interfaces

5.4.1 Notifications

At its configurable interfaces the J1939Nm defines notifications that can be mapped to callback functions provided by other modules. The mapping is not statically defined by the J1939Nm 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.

5.4.1.1 <User_AddressClaimedIndication>

Prototype	
`void` **`<User_AddressClaimedIndication>`** `(NetworkHandleType channel, uint8 sourceAddress, const uint8 *name)`	
Parameter	
`channel`	Channel on which the AC was received or transmitted.
`sourceAddress`	Claimed address.
`name`	NAME of the node that sent the AC.
Return code	
`void`	
Functional Description	
Provides the content of all received and all transmitted AddressClaimed (AC) messages.	
Particularities and Limitations	
none	
Call context	
May be called from interrupt or task level.	

Table 5-27 <User_AddressClaimedIndication>

6 Configuration

6.1 Configuration Variants

The J1939Nm supports the configuration variants

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

The configuration classes of the J1939Nm parameters depend on the supported configuration variants. For their definitions please see the J1939Nm_bswmd.arxml file.

6.2 Post-Build Configuration

The configuration of post-build loadable is described in [6].

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
AC	AddressClaimed (PGN 0x0EE00)
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
BSW	Basis Software
CA	CommandedAddress (PGN 0xFED8)
DEM	Diagnostic Event Manager
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)
SRS	Software Requirement Specification
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