

MICROSAR Classic FlexRay Network Management

Technical Reference

Nm_Asr4NmFr

Version 7.02.00

Document Information

History

Author	Date	Version	Remarks
vissrk	2012-07-31	1.00.00	ESCAN00058397 Creation
vismdr	2012-11-28	1.01.00	ESCAN00060628 Updated chapter 2.11.1.1 ESCAN00063145 Updated MICROSAR Architecture Overview Figure in chapter 1.2.1
vismdr	2013-05-11	1.02.00	ESCAN00064966 Support of Variant Post-Build-Loadable ESCAN00067279 Adapted Table 2-2, Removed 'Debugging Support' from chapter 2.1.3 ESCAN00067281 Merged chapter 'AUTOSAR Standard Compliance' with chapter 2, removed chapter 'Compiler Abstraction and Memory Mapping', various improvements ESCAN00067282 Added chapter 2.1.1.4, adapted chapter 4.2.1.1 ESCAN00067283 Updated Figure 2-1
vismdr	2013-10-01	2.00.00	ESCAN00067702 Added 'Runtime Measurement Support' to 'Features Beyond the AUTOSAR Standard' ESCAN00070809 Updated Figure 1-1
vismdr	2014-06-17	2.01.00	ESCAN00071157 Added chapter 2.6.8, adapted chapter 4.2.1.3 ESCAN00075261 Adapted chapter 2.4.3.3 ESCAN00075266 Adapted chapter 3.4, added chapter 3.4.1
vismdr	2014-10-07	3.00.00	ESCAN00076760 Adapted chapters 'Component History', 1, 2.1 ESCAN00078798 Updated Figure 1-1 ESCAN00078801 Updated chapter 4.2.2.1
vissrk	2015-04-14	4.00.00	ESCAN00082406 Updated Figure 1-1 ESCAN00082407 Updated Table 2-1 ESCAN00082307 Added chapter 2.6.6, created chapter 2.16
vissrk	2015-11-18	5.00.00	ESCAN00085698 Adapted chapter 3.4 ESCAN00086901 Adapted chapter 2.4.3.1 ESCAN00087413 Adapted chapter 2.1.1
vissrk	2016-03-28	6.00.00	No changes in this file
vissrk	2016-04-28	6.00.01	ESCAN00089769 Adapted chapter 2.11.1
vissrk	2016-06-17	6.00.02	ESCAN00086613 Adapted chapter 2.11.1

vispkn	2019-04-10	7.00.00	STORYC-8146 Added chapter 2.15.6, Modified chapters 2.6.3, 2.15
vispps	2022-03-24	7.01.00	NMM-1847 Added chapters 2.15.7, 4.2.2.13, adapted chapters 2.1, 2.1.1, 2.6.3, 2.11.1, 3.6
vispps	2022-06-23	7.02.00	Product name updated to MICROSAR Classic

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	Requirements on FlexRay	R4.0.3
[2]	AUTOSAR	Specification of FlexRay Network Management	R4.0.3
[3]	AUTOSAR	Specification of FlexRay Network Management	R21-11
[4]	AUTOSAR	Specification of Development Error Tracer	R4.0.3
[5]	AUTOSAR	Specification of Diagnostic Event Manager	R4.0.3
[6]	AUTOSAR	List of Basic Software Modules	R4.0.3
[7]	AUTOSAR	Specification of RTE	R4.0.3
[8]	Vector	Technical Reference MICROSAR Classic Network Management Interface	See delivery
[9]	Vector	Technical Reference MICROSAR Classic FlexRay Interface	See delivery
[10]	Vector	Technical Reference MICROSAR Classic FlexRay State Manager	See delivery

Scope of the Document

This technical reference describes the specific use of the FlexRay Network Management basic software.



Caution

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

Contents

1	Introduction.....	10
1.1	Naming Conventions.....	10
1.2	Architecture Overview.....	11
1.2.1	Architecture of AUTOSAR Software	11
1.2.2	Architecture of AUTOSAR Network Management.....	11
2	Functional Description.....	13
2.1	Features	13
2.1.1	Deviations Against AUTOSAR 4.0.3.....	14
2.1.1.1	RAM Initialization	14
2.1.1.2	Additional Configuration Dependencies	14
2.1.1.3	Cycle Counter Emulation Feature	14
2.1.1.4	Syntax of FrNm_Init	15
2.1.2	Additions/ Extensions.....	15
2.1.2.1	Single Channel Optimization.....	15
2.1.2.2	Memory Initialization	15
2.1.2.3	Immediate Transmission Only.....	15
2.1.2.4	NM Sync PDUs.....	15
2.1.3	Limitations.....	16
2.1.3.1	PDU Schedule Variants in one Cluster.....	16
2.1.3.2	Supported PDU Schedule Variants	16
2.1.3.3	FlexRay NM Main Function Handling.....	16
2.1.3.4	Identification of NM Messages	17
2.2	Network Management Mechanism.....	17
2.3	Initialization	19
2.4	Operational Modes and States.....	19
2.4.1	Bus-Sleep Mode	20
2.4.2	Synchronize Mode	20
2.4.3	Network Mode.....	20
2.4.3.1	Repeat Message State	21
2.4.3.2	Normal Operation State	22
2.4.3.3	Ready Sleep State	22
2.4.4	Bus-Communication Handling	23
2.5	Synchronization Loss Handling	24
2.5.1	Short-Term Loss Handling	24
2.5.2	Long-Term Loss Handling	25
2.6	Network Management Message Transmission and Reception	26
2.6.1	AUTOSAR FlexRay Interface.....	26
2.6.2	PDU Schedule Variant	26

2.6.3	PDU Message Layout	27
2.6.4	Decoupled Transmissions	29
2.6.5	NM Hardware Vector Support	29
2.6.6	Dual Channel Support.....	30
2.6.7	User Data.....	30
2.6.8	Voting Cycle and Data Cycle.....	31
2.6.8.1	Examples.....	31
2.7	Node Detection	32
2.8	NM PDU Receive Indication.....	33
2.9	Gateway Functionality.....	33
2.9.1	Remote Sleep Indication and Cancellation.....	33
2.9.2	Bus Synchronization	33
2.10	Coordinator Synchronization Support.....	34
2.11	Error Handling.....	34
2.11.1	Development Error Reporting.....	34
2.11.1.1	Parameter Checking	35
2.11.2	Production Code Error Reporting	36
2.12	Com User Data Support.....	36
2.12.1	Configuration Preconditions for AUTOSAR ECU Configurations	36
2.13	Active Wake-up Bit.....	37
2.14	Car Wake-up.....	37
2.14.1	Rx Path	38
2.14.2	Tx Path	38
2.15	Partial Networking.....	38
2.15.1	Availability of Partial Network Request Information	38
2.15.2	Transmission of the CRI Bit.....	38
2.15.3	Extended Handling for Received NM Messages	39
2.15.4	Aggregation of Requested Partial Networks	39
2.15.5	Spontaneous Sending of NM Messages	39
2.15.6	Transmission of CRI bit without Partial Networking	40
2.15.7	Synchronized PNC Shutdown	40
2.15.7.1	Synchronized PNC Shutdown Restrictions	40
2.16	Sync PDU Master in Dual Channel Use Case	40
2.16.1	Example Scenario.....	41
2.16.2	Sync PDU Master Algorithm.....	42
3	Integration.....	46
3.1	Scope of Delivery.....	46
3.1.1	Static Files	46
3.1.2	Dynamic Files	46
3.2	Include Structure.....	47

3.3	Version Changes.....	47
3.4	Main Function	47
3.4.1	OS Usage	49
3.5	Critical Sections	51
3.6	Critical Section Codes.....	51
4	API Description.....	54
4.1	Data Types.....	54
4.2	Services Provided by FRNM	54
4.2.1	Administrative Functions	54
4.2.1.1	FrNm_Init: Initialization of FlexRay NM	54
4.2.1.2	FrNm_InitMemory	55
4.2.1.3	FrNm_MainFunction: Channel-Specific Main Functions of FlexRay NM.....	55
4.2.2	Service Functions	56
4.2.2.1	FrNm_GetVersionInfo: Version Information API	56
4.2.2.2	FrNm_GetState: Get the State of the Network Management.....	57
4.2.2.3	FrNm_PassiveStartUp: Network Management Wake- Up.....	58
4.2.2.4	FrNm_NetworkRequest	59
4.2.2.5	FrNm_NetworkRelease.....	60
4.2.2.6	User Data Handling.....	61
4.2.2.6.1	FrNm_SetUserData: Set User Data	61
4.2.2.6.2	FrNm_GetUserData: Get User Data	62
4.2.2.6.3	FrNm_GetPduData: Get NM PDU Data	63
4.2.2.7	Node Detection	64
4.2.2.7.1	FrNm_RepeatMessageRequest: Set Repeat Message Request Bit	64
4.2.2.7.2	FrNm_GetNodentifier: Get Source Node Identifier	65
4.2.2.7.3	FrNm_GetLocalNodentifier: Get Local Source Node Identifier.....	66
4.2.2.8	Remote Sleep Indication.....	67
4.2.2.8.1	FrNm_CheckRemoteSleepIndication: Check for Remote Sleep Indication	67
4.2.2.9	Bus Synchronization	68
4.2.2.9.1	FrNm_RequestBusSynchronization: Synchronization of Networks.....	68
4.2.2.10	NM Message Transmission Request.....	69
4.2.2.10.1	FrNm_Transmit: Spontaneous NM Message Transmission	69
4.2.2.11	Communication Control Service.....	70

4.2.2.11.1	FrNm_EnableCommunication	70
4.2.2.11.2	FrNm_DisableCommunication	71
4.2.2.12	Coordinator Synchronization Support	72
4.2.2.12.1	FrNm_SetSleepReadyBit.....	72
4.2.2.13	Synchronized PNC Shutdown Support	73
4.2.2.13.1FrNm_RequestSynchronizedPncShutdown	73
4.3	Services used by FRNM	73
4.4	Callback Functions.....	75
4.4.1	Callback Functions from FlexRay Interface	75
4.4.1.1	FrNm_TxConfirmation: NM PDU Transmission Confirmation	75
4.4.1.2	FrNm_RxIndication: NM PDU Reception Indication	75
4.4.1.3	FrNm_TriggerTransmit: NM PDU pre-transmit function ..	76
4.4.2	Callback Function from FlexRay State Manager	77
4.4.2.1	FrNm_StartupError: FlexRay Synchronization Loss Indication	77
5	Glossary and Abbreviations	78
5.1	Glossary	78
5.2	Abbreviations	78
6	Contact.....	80

Illustrations

Figure 1-1	AUTOSAR 4.x Architecture Overview	11
Figure 1-2	Interfaces to adjacent modules of the FRNM	12
Figure 2-1	FlexRay NM state diagram according to the FlexRay NM SWS [2]	18
Figure 2-2	Example for the transition from Ready Sleep to Bus Sleep	23
Figure 2-3	Example for Data Cycle = 8 with five nodes sending their NM PDUs	31
Figure 2-4	Example for Data Cycle = 4 with five nodes where one node eventually cannot send its NM PDU	32
Figure 2-5	Example for Node 1 having 'Data Cycle' 2 and Nodes 2 and 3 having 'Data Cycle' 4	32
Figure 2-6	Example scenario for Sync PDU Master usage	41
Figure 2-7	NM Sync PDU Master Algorithm	43
Figure 2-8	Different scenarios for Sync PDU Master	45
Figure 3-1	Include structure	47
Figure 3-2	FlexRay NM Task Execution	48
Figure 3-3	Task Activation through Relative Alarms	49
Figure 3-4	Task Activation through Relative Alarms if FrNm task is not the first task scheduled by SetRelAlarm.....	50
Figure 3-5	Task Activation in Job List Context.....	50

Tables

Table 1-1	Naming Conventions	10
Table 2-1	Supported AUTOSAR standard conform features	13
Table 2-2	Not supported AUTOSAR standard conform features	14
Table 2-3	Features provided beyond the AUTOSAR standard	15
Table 2-4	Default PDU Message Layout.....	27
Table 2-5	Separate NM Vote Layout.....	27
Table 2-6	Layout of the Control Bit Vector	27
Table 2-7	PDU length value table	28
Table 2-8	Number of Tx-PDUs needed.....	30
Table 2-9	Service IDs	35
Table 2-10	Errors reported to DET	35
Table 2-11	Errors reported to DEM.....	36
Table 2-12	Configuration Precondition Overview for AUTOSAR ECU Configurations .	37
Table 3-1	Static files	46
Table 3-2	Generated files	46
Table 3-3	Critical Section Codes	53
Table 4-1	FrNm_Init.....	54
Table 4-2	FrNm_InitMemory	55
Table 4-3	FrNm_MainFunction	56
Table 4-4	FrNm_GetVersionInfo	56
Table 4-5	FrNm_GetState	57
Table 4-6	FrNm_PassiveStartUp	58
Table 4-7	FrNm_NetworkRequest	59
Table 4-8	FrNm_NetworkRelease	60
Table 4-9	FrNm_SetUserData	61
Table 4-10	FrNm_GetUserData.....	62
Table 4-11	FrNm_GetPduData	63
Table 4-12	FrNm_RepeatMessageRequest.....	64
Table 4-13	FrNm_GetNodeIdentifier.....	65
Table 4-14	FrNm_GetLocalNodeIdentifier	66

Table 4-15	FrNm_CheckRemoteSleepIndication	67
Table 4-16	FrNm_RequestBusSynchronization	68
Table 4-17	FrNm_Transmit	69
Table 4-18	FrNm_EnableCommunication	70
Table 4-19	FrNm_DisableCommunication	71
Table 4-20	FrNm_SetSleepReadyBit	72
Table 4-21	FrNm_RequestSynchronizedPncShutdown	73
Table 4-22	Services used by the FRNM	74
Table 4-23	FrNm_TxConfirmation	75
Table 4-24	FrNm_RxIndication	76
Table 4-25	FrNm_TriggerTransmit	76
Table 4-26	FrNm_StartupError	77
Table 5-1	Glossary	78
Table 5-2	Abbreviations	79

1 Introduction

This document describes the functionality, API and configuration of the AUTOSAR BSW module FRNM as specified in [2]. Also the integration of the FlexRay Network Management into the AUTOSAR stack is covered by this document.

Neither the CAN Network Management, nor the LIN Network Management, nor the UDP Network Management is covered by this document.

Please note that in this document the term Application is not used strictly for the user software but also for any higher software layer, like e.g. the AUTOSAR Network Management Interface. Therefore, Application refers to any of the software components using the FlexRay NM.

For further information please also refer to the AUTOSAR SWS specifications referenced in chapter ‘Reference Documents’.

Supported AUTOSAR Release*:	4	
Supported Configuration Variants:	pre-compile, post-build-loadable, post-build-selectable	
Vendor ID:	FRNM_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	FRNM_MODULE_ID	32 decimal (according to ref. [6])

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

1.1 Naming Conventions

The names of the service functions provided by the FlexRay NM always start with a prefix that denominates the software component where the service is located. E.g. a service that starts with ‘FrNm_’ is implemented within the FlexRay NM.

Naming conventions	
Nm_	Services of NM Interface
FrNm_	Services of FlexRay NM
Frlf_	Services of FlexRay Interface
Det_	Services of Development Error Tracer

Table 1-1 Naming Conventions

Nodes which are configured to be passive will be also referred to as passive nodes. Accordingly nodes that are not passive will be termed as active nodes. The default for this document is an active node. Differences to a passive node are described in the corresponding chapters.

The term ‘NM message’ is used synonymously to ‘NM PDU’.

1.2 Architecture Overview

1.2.1 Architecture of AUTOSAR Software

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

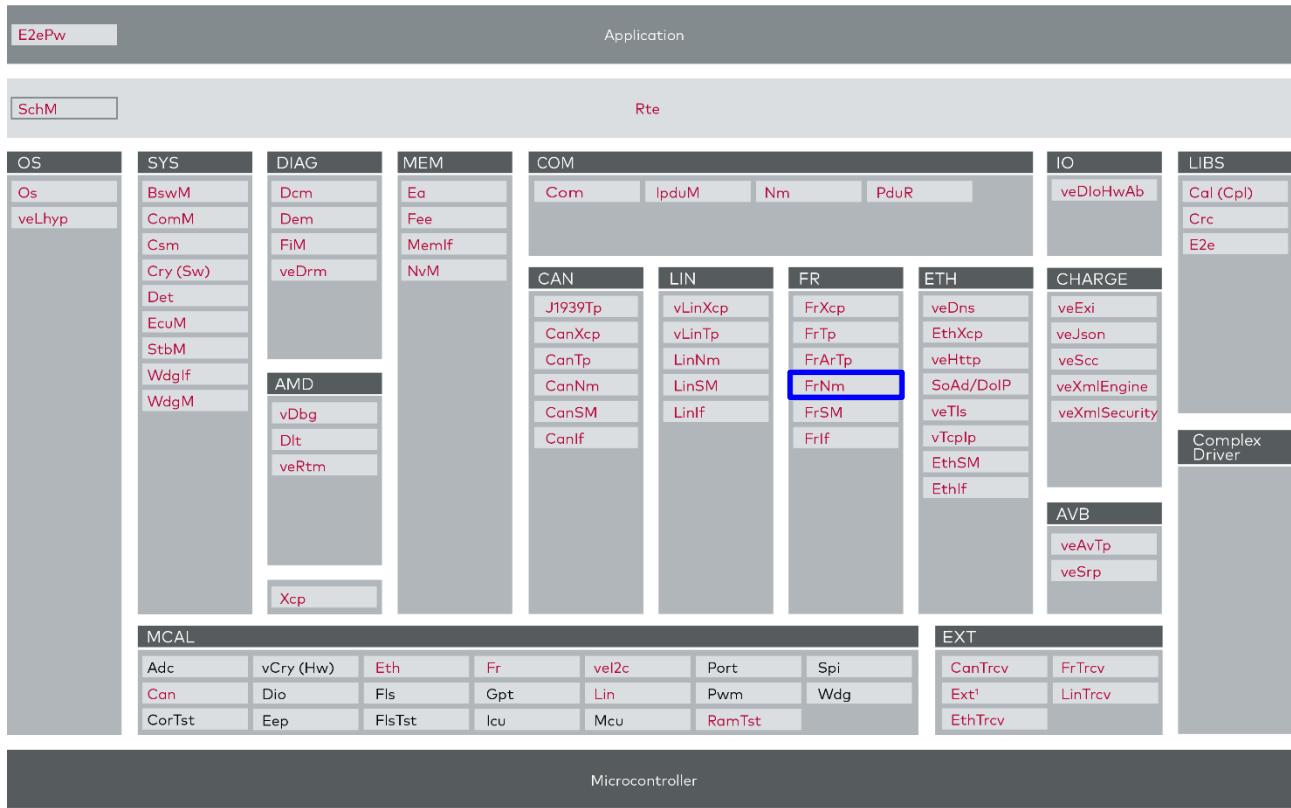


Figure 1-1 AUTOSAR 4.x Architecture Overview

1.2.2 Architecture of AUTOSAR Network Management

In the current AUTOSAR Release the standard AUTOSAR Network Management may consist of up to five modules:

- > NM Interface¹
- > CAN NM¹
- > FlexRay NM
- > LIN NM¹
- > UDP NM¹

The NM Interface schedules function calls from the application to the respective module for each NM channel, e.g. for a FlexRay NM cluster the corresponding FlexRay NM function will be called.

The communication bus specific functionality is incorporated in the corresponding bus-specific NM. For FlexRay this is the FlexRay NM.

¹ Not covered by this document

The next figure shows the interfaces to adjacent modules of the FlexRay NM. These interfaces are described in chapter 4 ‘API Description’.

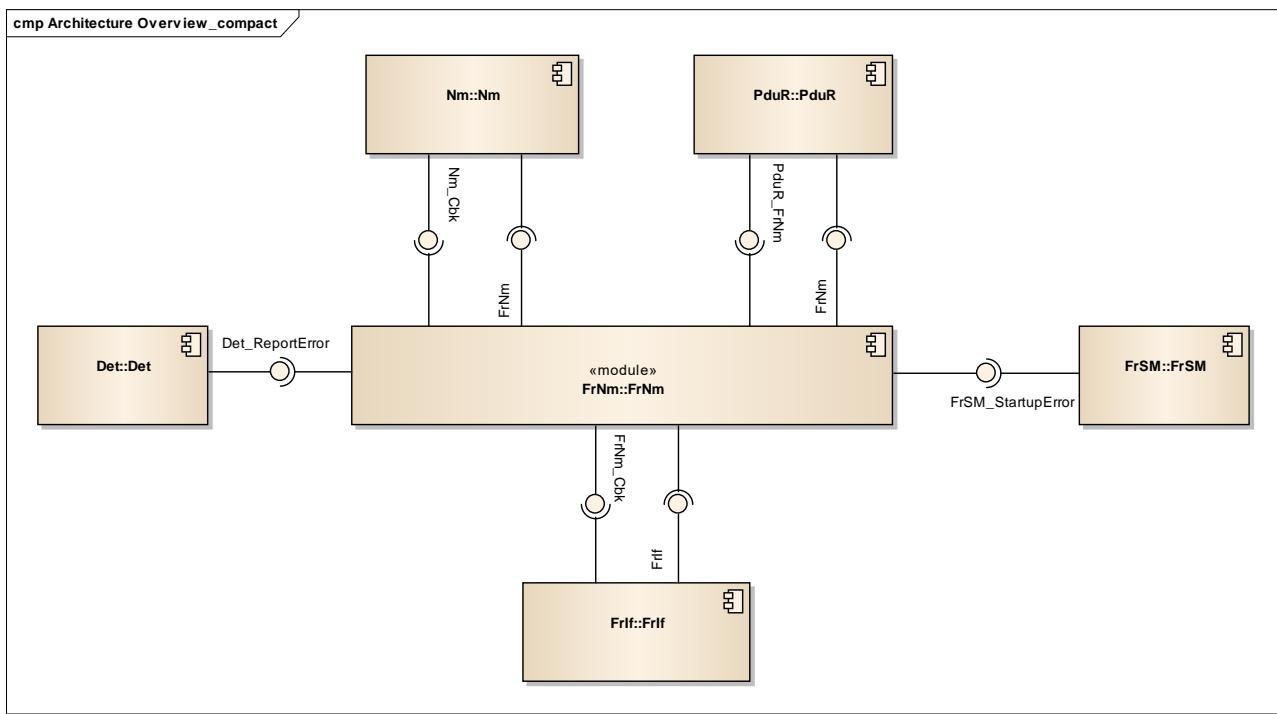


Figure 1-2 Interfaces to adjacent modules of the FRNM

2 Functional Description

2.1 Features

The Network Management is a network comprehensive protocol that provides services for the organization of the network. It is a decentralized and direct Network Management, meaning every ECU transmits a special network management message, which is reserved for the Network Management only.

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

The AUTOSAR standard functionality is specified in [2], the corresponding features are listed in the tables

- > Table 2-1 Supported AUTOSAR standard conform features
- > Table 2-2 Not supported AUTOSAR standard conform features

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

- > Table 2-3 Features provided beyond the AUTOSAR standard

The following features specified in [2] are supported:

Supported AUTOSAR Standard Conform Features
Controlled transition of all ECU's to bus-sleep mode and vice versa.
Periodic update of FlexRay NM messages
User Data Handling
Remote Sleep Indication
Node Detection
Com User Data Support
Passive Mode
PDU Rx Indication
State Change Notification
Dual Channel PDU Support
Car Wake-up
Coordinator Sync Support
Partial Networking
Transmission Error Handling
Active Wake-up Bit
Post-Build Loadable
MICROSAR Classic Identity Manager using Post-Build Selectable
Synchronized PNC Shutdown (specified in [3])

Table 2-1 Supported AUTOSAR standard conform features

2.1.1 Deviations Against AUTOSAR 4.0.3

The deviations of FlexRay NM from the AUTOSAR specifications [1] and [2] are provided in this section. The following features specified in [2] are not supported:

Category	Description	ASR Version
Functional	Schedule details [ch. 7.9] PDU Schedule Variants 3–7 are not supported	4.0.3
Functional	FlexRay NM-Data PDU format [ch. 7.6.2.1] FRNM324 is not fulfilled. In case ControlBitVectorActive is OFF, the first byte is used as User Data in case ScheduleVariant is not 1.	4.0.3
Functional	The MICROSAR Classic FrNm does not support reliable TxConfirmation. Therefore it can occur that the synchronized shutdown cannot be executed. For more details refer to chapter 2.15.7.1.	R21-11

Table 2-2 Not supported AUTOSAR standard conform features

Other deviations are provided as follows.

2.1.1.1 RAM Initialization

If RAM is not implicitly initialized at the start-up, the function `FrNm_InitMemory` has to be called.

2.1.1.2 Additional Configuration Dependencies

Following additional dependency between configuration parameters are added to avoid bad configurations:

- > Node Detection cannot be enabled if Node Identifier is disabled.
- > Node Detection cannot be enabled if Control Bit Vector is disabled.
- > Repeat Message Bit has to be always set to the same value as Node Detection

2.1.1.3 Cycle Counter Emulation Feature

The Cycle Counter Emulation feature is implemented according to a proposal to be implemented in the next version of [2], discussed in AUTOSAR Bugzilla entry 52552. This includes the following behavior if the feature is enabled:

- > There is no Synchronization Loss Timer realized by an OS timer. Instead, the FlexRay Cycle Counter is emulated during synchronization loss by just incrementing the previous Cycle Counter value inside the `FrNm_MainFunction_X` functions. This assumes that `FrNm_MainFunction_X` is continued to be called in the same intervals as before a synchronization loss has occurred.
- > The Ready Sleep Counter is decremented during synchronization loss by using the simulated FlexRay Cycle Counter to determine the end of a repetition cycle.
- > The configuration parameter `FrNmSyncLossTimer` is not used.
- > The transition to Bus Sleep Mode due to a call of `FrNm_StartupError()` is only performed if the current state is Synchronize and the network is not requested.

- > The transition to Bus Sleep Mode is not triggered from Repeat Message if the global time cannot be retrieved.

2.1.1.4 Syntax of FrNm_Init

The function prototype of FrNm_Init (see also chapter 4.2.1.1) deviates from the prototype provided in [2]. The function prototype provided by the FrNm implementation has an additional const classifier for the nmConfigPtr parameter.

2.1.2 Additions/ Extensions

The following extensions of the FlexRay NM software specifications ([2]) are available within the Network Management embedded software components. If required, the extensions have to be enabled during configuration.

Features Provided Beyond The AUTOSAR Standard
Single Channel Optimization
Memory Initialization
Immediate Transmission Only
Runtime Measurement Support
Sync PDU Master Dual Channel

Table 2-3 Features provided beyond the AUTOSAR standard

2.1.2.1 Single Channel Optimization

For single channel systems it is possible to optimize the source code for saving precious resources (ROM, RAM and CPU load). This optimization is only possible when source code is available.

Please note that single channel optimization can only be enabled in pre-compile configurations.

2.1.2.2 Memory Initialization

Not every start-up code of embedded targets and neither CANoe reinitialize all variables correctly. It thus may happen that the state of a variable may be not initialized, when this should be the case. It therefore needs to be initialized explicitly by the corresponding module.

Refer also to chapter 4.2.1.2 ‘FrNm_InitMemory’ for this additional functionality.

2.1.2.3 Immediate Transmission Only

In case that all transmissions of NM messages are configured as immediate, it is possible to optimize the source code. Please note that this optimization can only be set for pre-compile configurations. If this value is enabled all NM messages must be transmitted immediately.

2.1.2.4 NM Sync PDUs

The FlexRay NM supports the possibility to send ‘NM Sync PDUs’ on a node in a cluster where ‘Dual Channel PDU Enabled’ is ON in Dual Channel setups. This is used to synchronize awake requests between the two physical FlexRay channels. Details can be found in chapter 2.16 ‘Sync PDU Master in Dual Channel Use Case’.

2.1.3 Limitations

2.1.3.1 PDU Schedule Variants in one Cluster

Due to the reception handling in cases of different PDU Schedule Variants in one cluster is not specified clearly in [2] and a solution therefore would need runtime and memory space the usage of the PDU Schedule Variant is restricted to be the same for all ECUs in one cluster.

Please note that this is also recommended by AUTOSAR (see [2]).

2.1.3.2 Supported PDU Schedule Variants

Currently the PDU Schedule Variant 1 and 2 are supported.

2.1.3.3 FlexRay NM Main Function Handling

Since the NM task has to be synchronized with the FlexRay cycle and can only be activated in a specific time window (refer to chapter 3.4 'Main Function') some restrictions apply:

- > The task activation window has to be wide enough for a complete NM task run
 - > Depending on the configuration a NM task may have different runtimes. Therefore an upper limit for the used configuration has to be known at integration time.



Caution

Since the FlexRay NM task locks interrupts only where needed, it can be interrupted. Possible interruptions of the NM task have to be considered for the upper limit of the runtime.

- > Additionally an upper limit for the duration of receiving and transmitting NM messages over the FlexRay Interface has to be taken into account, because they influence the activation window of the main function.
- > If the NM Hardware Vector Support (FRNM_HW_VOTE_ENABLED) is used it has to be taken into account at which point in time the aggregated information is available which also influences the start point of the main function activation window.
- > If decoupled transmission mode is used for NM messages this leads to higher transmission duration.[9]
- > Depending on these timing values restrictions for the placement of NM messages in the FlexRay schedule may apply since it must be ensured that the own NM messages can be transmitted and all necessary NM Information can be received.
- > In difference to the description in [2] the left border of the activation window does not only depend on the NM Votes that must be received, but also on the Control Bit Vector information of all NM Nodes due to this information can also influence the state decision (refer to chapter 2.7 'Node Detection').

- > If the activation of the NM task is done at the end of a FlexRay cycle it has to be ensured that the task execution does not cross the cycle since otherwise it cannot be ensured that the FlexRay cycle is read out from the FlexRay Interface consistently.
- > If the activation of the NM task is only possible after a new FlexRay cycle has started, the NM task should be executed as soon as possible. Otherwise the NM messages are restricted to frames back in the cycle.
- > The FlexRay NM needs to know whether it is executed shortly before cycle end or shortly after cycle start. Therefore this has to be configured in the tool.

**Caution**

These restrictions must be strictly followed. Otherwise the FlexRay Network Managements in the cluster may behave inconsistent regarding their timings and state transitions.

2.1.3.4 Identification of NM Messages

Since with FIBEX file formats < 3.x only a frame and not a PDU can be marked to be from type NM for the identification of a NM message by the generation tool it is necessary to restrict a NM frame to hold only one NM PDU and no further messages. In the static segment this may lead to a waste of bandwidth.

When NM Vote and NM Data are transmitted in separate messages the generation tool must be able to distinguish between them. Therefore the naming of the separate NM Data PDU has to contain the keyword 'DATA' in its name. The keyword can occur at an arbitrary position in the name, e.g. as postfix. For a better readability it is recommended to separate the keyword with underscores.

2.2 Network Management Mechanism

As described above the AUTOSAR NM is a decentralized direct Network Management. This means that every network node has the same functionality and performs state and operation mode changes self-sufficiently depending on the internal state and whether the bus-communication is required by some node or not.

The network management mechanism is quite simple:

- > Every network node votes for keeping the bus awake only as long as it needs to communicate with other network nodes. Normally bus-communication is required as long as clamp 15 is set (ignition is turned on) or during follow-up. Voting is done in the static FlexRay segment by setting the vote bit. In the dynamic segment the presence of the NM (vote) PDU corresponds to the vote.
- > If there is no more network node in the whole network that needs to communicate with other network nodes, all nodes stop voting for keeping the bus awake. For the static segment this means that the vote bit is cleared. In the dynamic segment no more NM (vote) PDU is transmitted.

- > Each network node performs a transition to bus-sleep mode after some time has passed since the last repetition cycle, because no vote for keeping the bus awake has occurred. Therefore all nodes will go to bus-sleep mode together.



FAQ

The application is in charge of the decision whether the bus-communication is required or not.

The following figure shows the FlexRay NM state diagram with all state transitions, the corresponding conditions and actions:

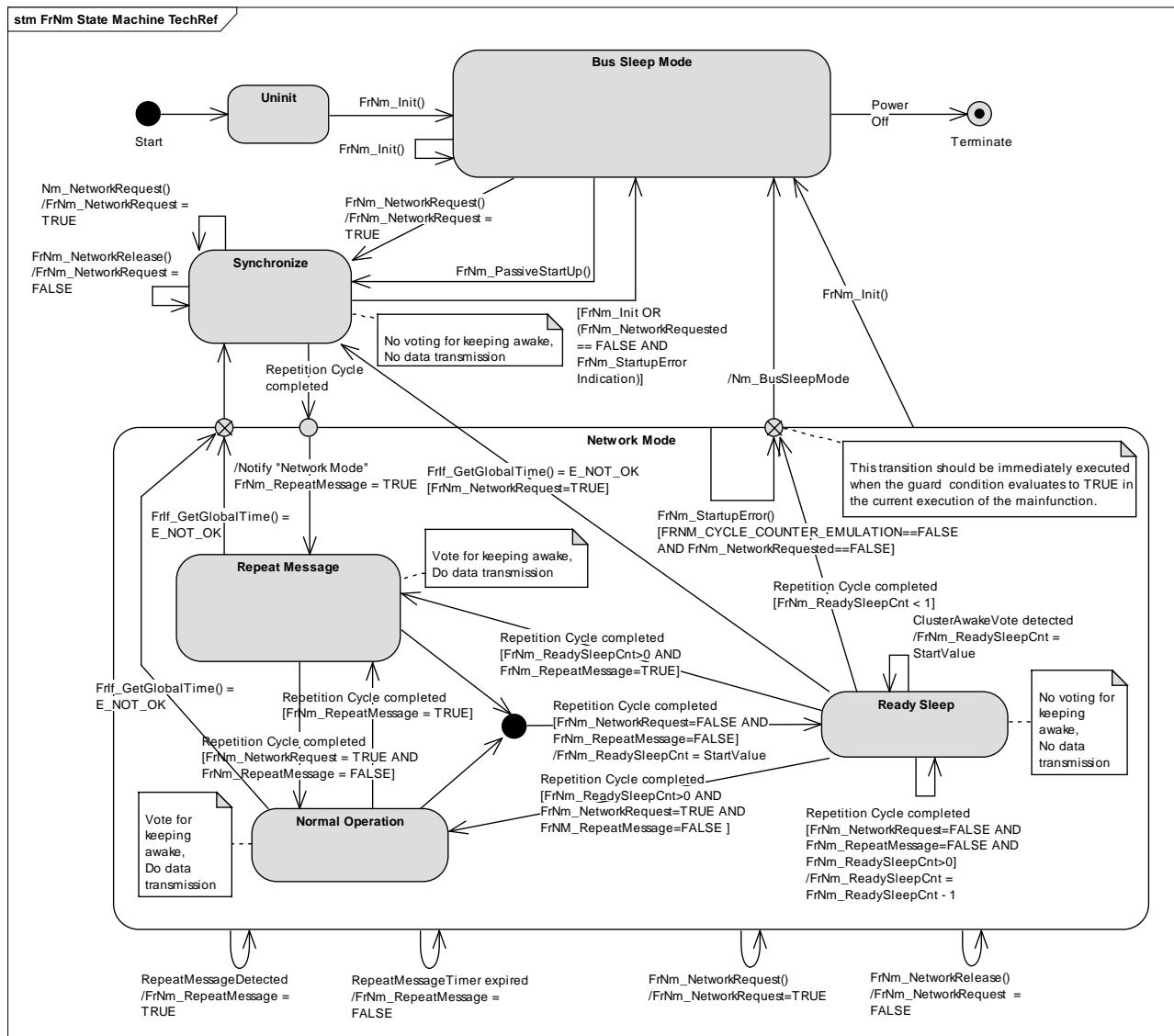


Figure 2-1 FlexRay NM state diagram according to the FlexRay NM SWS [2]

Since the FlexRay NM has to run synchronously to the FlexRay cycle almost all state transitions are only possible at the end of a repetition cycle. The duration of a repetition cycle is a configurable number of FlexRay cycles.

2.3 Initialization

Before the FlexRay NM can be used it has to be initialized by the application. The initialization has to be carried out before any other functionality of the FlexRay NM is executed. It shall take place after initialization of the FlexRay Interface and prior to initialization of the NM Interface.

Also refer to chapter 4.2.1.1 ‘FrNm_Init: Initialization of FlexRay NM’.



Caution

The FlexRay NM assumes that some variables are initialized with zero at start-up. If the embedded target does not initialize RAM within the start-up code the function ‘FrNm_InitMemory’ has to be called during start-up and before the initialization is performed. Refer also to chapter 2.1.2.2 ‘Memory Initialization’.



Note

In an AUTOSAR environment where the ECU Manager and/or BSW Manager are used, the initialization is performed within the ECU Manager/BSW Manager.

If the variant Post-Build Selectable is used, multiple configurations exist and depending on which pre-defined variant shall be active, the correct one has to be chosen.

2.4 Operational Modes and States

The AUTOSAR NM consists of three operational modes:

- > Bus-Sleep Mode
- > Synchronize Mode
- > Network Mode

The NM Interface is notified about changes of the operation mode by calling the following functions:

Entering Network Mode:

`Nm_NetworkMode ()` (4.3)

Leaving Network Mode:

`Nm_BusSleepMode ()` (4.3)

Information about the current state and the current mode is provided by the service call:

`FrNm_GetState ()` (4.2.2.2)

The NM Interface is notified about changes of the states by the optional function:

`Nm_StateChangeNotification ()` (4.3)

2.4.1 Bus-Sleep Mode

The Bus-Sleep Mode is entered directly after initialization of the FrNm module. The Module stays in this state until a wakeup event is signaled by the application. There are two possible reasons for waking up:

If a node on the FlexRay bus demands communication then Bus-Sleep Mode is left by a call of:

`FrNm_PassiveStartUp()` (4.2.2.3)

Alternatively the network is requested by the own node which leads to a call of:

`FrNm_NetworkRequest()` (4.2.2.4)

In both cases Synchronize Mode will be entered (see chapter 2.4.2 ‘Synchronize Mode’).

If a NM message is received in Bus-Sleep Mode the service

`Nm_NetworkStartIndication()` (4.3)

is called by FlexRay NM.

All network nodes perform a transition from Network Mode to Bus-Sleep Mode at almost the same time. The transition to this mode is bound to the end of a repetition cycle.

2.4.2 Synchronize Mode

This mode synchronizes the wake-up of FlexRay NM to the FlexRay cycle, i.e. this state is automatically left after the current repetition cycle is completed and the FlexRay Interface is in online mode.

2.4.3 Network Mode

The Network Mode comprises three states:

- > Repeat Message
- > Normal Operation
- > Ready Sleep

Within this mode the ECU is ‘online’ and participates in the network communication. The three states describe the overall behavior of the Module:

- > Active participation: the node does request bus communication (Repeat message State and Normal Operation State).
- > Passive participation: the node does not request bus communication (Ready Sleep State) and another node keeps the network awake.

**Note**

Nodes configured as passive (FRNM_PASSIVE_MODE_ENABLED = TRUE) cannot enter Normal Operation State and should when entering Repeat Message State immediately transit to Ready Sleep (this can be achieved by configuring the Repeat Message Time to zero).

When the Network Management enters the Network Mode the application is notified by a call of the function:

Nm_NetworkMode () (4.3)

When the Network Management leaves the Network Mode the application is notified by a call of the function:

Nm_BusSleepMode () (4.3)

**Note**

All state transitions in the Network Mode are bound to the end of a repetition cycle.

**Note**

The applications communication is active during Network Mode. It is started upon entry and stopped upon exit of Network Mode. I.e. application messages are transmitted and received within Network Mode!

2.4.3.1 Repeat Message State

The Repeat Message State is entered from Synchronize Mode at the end of the current repetition cycle. In this state the NM votes for keeping the bus awake.

The Repeat Message State is left after a certain configurable time.

It is recommended that the setting of the Repeat Message Time should be equal for all nodes on the network, because all state transitions are done synchronously on all nodes.

It is also possible to set Repeat Message Time to 0.

**Note**

When FrNm is in Repeat Message, Sending / Receiving of NM votes has no influence on the shutdown behavior of the ECU itself (in contradiction to CanNm/UdpNm).

Depending on the bus-communication needs of the application Normal Operation State or Ready Sleep State is entered upon exit of Repeat Message State. Since passive nodes do not participate actively in the network they cannot enter Normal Operation State.

2.4.3.2 Normal Operation State

This state can only be entered by active nodes.

The local bus-communication request of the application is distributed in the network by the transmission of positive NM votes.

The network management stays in Normal Operation State until the application releases the bus communication, which leads to a state change to Ready Sleep State.

2.4.3.3 Ready Sleep State

The network management stays in Ready Sleep State as long as the application does not request bus-communication and the application of at least one other node still requests bus-communication (by voting for keeping the bus awake).

When a certain configurable time (given in repetition cycles) after the last network node has released bus-communication (i.e. the last positive vote has been received) has passed, the transition to Bus-Sleep Mode is performed and Network Mode is left. This transition is only possible after the completion of a repetition cycle.

**Caution**

When entering Bus-Sleep Mode the physical bus interface has to be put in sleep mode. On FlexRay NM clusters the transceiver and the FlexRay-Controller have to be put in sleep mode.

On completion of the repetition cycle with the last positive NM Vote, it takes

$$('Ready Sleep Cnt' + 1) \times (\text{Repetition Cycle})$$

FlexRay Cycles until the transition to Bus Sleep Mode is triggered.

Figure 2-2 provides an example for *Repetition Cycle* being 2 ('Main Across Fr Cycle' is TRUE, see also chapter 3.4). The last positive NM Vote is received in FlexRay Cycle 0 in this example. The effect of this reception is that the Ready Sleep Timer is restarted in the context of the next FrNm Main Function call.

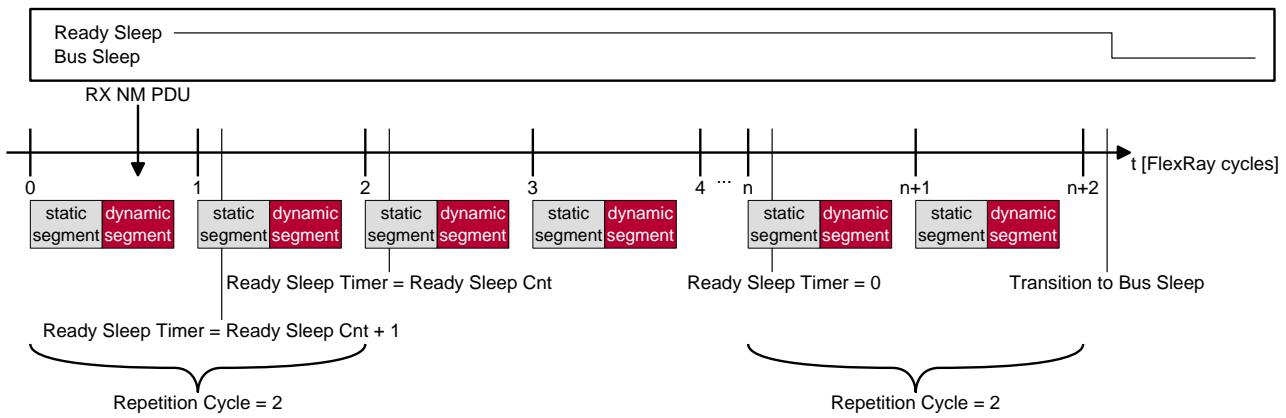


Figure 2-2 Example for the transition from Ready Sleep to Bus Sleep

After each Repetition Cycle Completion, the Ready Sleep Timer is decremented by one within the context of the FrNm Main Function. On the last Repetition Cycle Completion in FlexRay Cycle ($n+2$), the transition to Bus Sleep is triggered.

Let *Repetition Cycle* be 2, *Ready Sleep Cnt* be 2, the time for one *FlexRay Cycle* be 5 ms. Then the transition to Bus Sleep occurs $((2 + 1) \times 2 \times 5)$ ms = 30 ms after the Repetition Cycle Completion with the last positive vote reception.



Caution

There have been ambiguities in the AUTOSAR Specification [2] concerning the time it takes until the transition to Bus Sleep occurs after the last positive vote. These ambiguities have been corrected by AUTOSAR in more recent versions of [2]. A possible misinterpretation of the AUTOSAR requirements was that this time would be $(\text{Ready Sleep Cnt}) \times (\text{Repetition Cycle}) \times (\text{Flex Ray Cycle Time})$.

If the requirements for your ECU concerning the remaining time in Ready Sleep after the last positive NM Vote are based on this misinterpretation, just configure the 'Ready Sleep Cnt' setting to the original value minus one.

As a summary, the Vector realization of FrNm is closer to the AUTOSAR $\geq 4.1.1$ requirements concerning this aspect, which means that it takes

$(\text{Ready Sleep Cnt} + 1) \times (\text{Repetition Cycle}) \times (\text{Flex Ray Cycle Time})$

until the transition to Bus Sleep happens after the Repetition Cycle Completion with the last positive NM Vote.

2.4.4 Bus-Communication Handling

The network management needs to know whether the application requires bus-communication. By default the network management does not actively participate in the network. The active participation in the network is requested by the service

FrNm_NetworkRequest() (4.2.2.4)

Calling this function in Bus-Sleep Mode starts the network and leads to a transition to Synchronize Mode (see chapter 2.4.1 'Bus-Sleep Mode').

If bus-communication is not required anymore it can be released with the service

FrNm_NetworkRelease()

(4.2.2.5)

Since passive nodes cannot participate actively in the network these APIs are not available for them.

Note that a bus-communication request is handled within the next task. In Bus-Sleep mode it is ensured that a communication request always leads to start-up even if the communication is released before the next task is executed.

2.5 Synchronization Loss Handling

As almost all state transitions and actions of the FlexRay NM are synchronized to a repetition cycle, the NM cannot react properly if the FlexRay synchronization, i.e. the base for the repetition cycle computation, gets lost. Additionally during synchronization loss no vote information can be sent or received. Therefore the NM cannot guarantee a synchronized shutdown in such situations. The synchronization loss handlings, one for short-term and one for long-term losses, are described in the following two chapters.

The behavior of the FlexRay NM depends on the Parameter switch Cycle Counter Emulation.

**Note**

Note that no special handling is necessary when the FlexRay NM is in state Bus Sleep.

2.5.1 Short-Term Loss Handling

For handling short-term losses the FlexRay NM reacts immediately on the situation with a state transition to Synchronize Mode in the states Repeat Message and Normal Operation. The FlexRay NM stays in Synchronize Mode as long as the synchronization cannot be regained. This ensures that an immediate re-synchronization is possible.

Cycle Counter Emulation Off:

If the FlexRay NM is in Ready Sleep and a Network Request is pending a state transition to Synchronize Mode has to be performed to ensure that the network restarts when synchronization can be gained again.

Otherwise (Network not requested) no actions occur as the ECU is ready to sleep and either it can synchronize again and shuts down or shut down is forced because of a long-term synchronization loss.

Cycle Counter Emulation On:

An additional timer in the Ready Sleep State emulates the repetition cycle and leads to a shutdown at approximately the same time as the ECU would shut down under normal conditions.

2.5.2 Long-Term Loss Handling

If a synchronization loss cannot be recovered within a certain time it is assumed that the issue cannot be solved at the moment. Therefore the FlexRay State Manager will inform the FlexRay NM about this situation by calling the function

`FrNm_StartupError()` (4.4.2.1)

This function is called cyclically until the ECU shuts down or the situation could be recovered. If the FlexRay NM has no internal network request it will trigger an immediate transition to Bus Sleep to allow the ECU to shut down.

See [10] for further information on `FrNm_StartupError()`.

Cycle Counter Emulation Off:

In Ready Sleep State the FlexRay NM performs a transition to Bus Sleep State if the network is not requested (otherwise it changes to synchronize state due to the short-term synchronization loss handling).

Cycle Counter Emulation On:

An additional timer in the Ready Sleep State emulates the repetition cycle and leads to a shutdown at approximately the same time as the ECU would shut down under normal conditions (same handling as already done in case of a short-term synchronization loss).

2.6 Network Management Message Transmission and Reception

2.6.1 AUTOSAR FlexRay Interface

The network management requests the transmission of NM messages by calling the service `FrIf_Transmit` (refer to [9] for more information about this function). The application has to take care of the user data. For details refer to chapter 2.6.7 'User Data'.

The successful transmission of every network management message is confirmed by the FlexRay Interface with the service

`FrNm_TxConfirmation()` (4.4.1.1)

The FlexRay Interface indicates the reception of NM message by calling the service

`FrNm_RxIndication()` (4.4.1.2)

2.6.2 PDU Schedule Variant

FlexRay NM offers seven possibilities of transmitting and receiving the NM messages. These variants differ mainly in the occurrence in the static or dynamic segment and the combined or separate transmission of NM Vote and NM Data. Refer to chapter 2.6.3 for more details about the contents of 'NM Vote' and 'NM Data'.

These seven variants are described in the following list:

1. NM Vote and NM Data transmitted within one NM PDU in the static segment.
2. NM Vote and NM Data transmitted within one NM PDU in the dynamic segment.
3. NM Vote and NM Data transmitted in separate NM PDUs in the static segment.
4. NM Vote is transmitted within one NM PDU in the static segment and the NM Data is transmitted within one NM PDU in the dynamic segment.
5. NM Vote is transmitted within one NM PDU in the dynamic segment and the NM Data is transmitted within one NM PDU in the static segment.
6. NM Vote and NM Data transmitted in separate NM PDUs in the dynamic segment.
7. NM Vote is transmitted within one NM PDU in the static segment and the NM Data is transmitted within one NM PDU in the dynamic segment (as variant 4) and additionally the NM Vote contains a copy of the CBV.

Although AUTOSAR allows configuring every node in a cluster independently they recommend using a common PDU Schedule Variant. Please refer also to chapter 2.1.3.1 'PDU Schedule Variants in one Cluster'.

2.6.3 PDU Message Layout

The default PDU Message Layout is described in the following table:

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte n					User data n – 2			
Byte n-1					User data n – 3			
...					...			
...					...			
Byte 3					User data 1			
Byte 2					User data 0			
Byte 1					Source Node Identifier			
Byte 0	Vote Bit ²				Control Bit Vector ³			

Table 2-4 Default PDU Message Layout

This layout applies for the combined transmission of NM Vote and NM Data and for the separated transmission of NM Data. The PDU Length can be zero to 254, depending on the configuration. For PDU Schedule Variant 1 at least the Byte 0 (containing the vote bit) has to be available.

If the NM Vote is transmitted separately the following layout applies:

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0	Vote Bit ⁴				Control Bit Vector ⁵			

Table 2-5 Separate NM Vote Layout

For the static segment the PDU Length of the vote message has to be one as the vote bit must be always available, in the dynamic segment it can be zero or one.

The layout of the Control Bit Vector (CBV) is provided in the following table:

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0	Vote Bit	Cluster Request Information Bit	Reserved	Active Wakeup Bit	NM Coordinator Sleep Ready	Reserved	PNSR	Repeat Message Request

Table 2-6 Layout of the Control Bit Vector

The exact configuration of the NM PDU Message Layout depends on the PDU Schedule Variant and on the availability of the features Control Bit Vector, Source Node Identifier and User Data Support.

² Only present for PDU Schedule Variant 1, otherwise this bit in the CBV is blocked.

³ Only present if CBV is enabled. If CBV is disabled and PDU Schedule Variant is 1 these bits are blocked as the vote bit must be present in Bit 7 of this byte.

⁴ Only present in the static segment. In the dynamic segment this bit is not available or blocked.

⁵ CBV is only present in PDU Schedule Variant 7 as a copy of the CBV in the separated data (CBV support has to be enabled). Otherwise those bits are blocked or in the dynamic segment not available if the PDU Length of the vote message is 0.

The allowed minimum and maximum⁶ configurable PDU Length values (and the resulting minimum and maximum number of User Data Bytes) for the Data PDU are listed within the following table:

Schedule Variant	PDU Schedule Variant 1 Minimum / Maximum	PDU Schedule Variant 2 Minimum / Maximum	PDU Schedule Variant 3 Minimum / Maximum	PDU Schedule Variant 4 Minimum / Maximum	PDU Schedule Variant 5 Minimum / Maximum	PDU Schedule Variant 6 Minimum / Maximum	PDU Schedule Variant 7 Minimum / Maximum
Configuration Parameter							
Control Bit Vector == OFF							
Node Identifier == OFF	1 / 1 (0 / 0)	0 / 0 (0 / 0)					
User Data Support == OFF							
Control Bit Vector == OFF							
Node Identifier == OFF	2 / 254 (1 / 253)	1 / 254 (1 / 254)					
User Data Support == ON							
Control Bit Vector == OFF							
Node Identifier == ON	2 / 2 (0 / 0)	1 / 1 (0 / 0)					
User Data Support == OFF							
Control Bit Vector == OFF							
Node Identifier == ON	3 / 254 (1 / 252)	2 / 254 (1 / 253)					
User Data Support == ON							
Control Bit Vector == ON							
Node Identifier == OFF	1 / 1 (0 / 0)						
User Data Support == OFF							
Control Bit Vector == ON							
Node Identifier == OFF	2 / 254 (1 / 253)						
User Data Support == ON							
Control Bit Vector == ON							
Node Identifier == ON	2 / 2 (0 / 0)						
User Data Support == OFF							
Control Bit Vector == ON							
Node Identifier == ON	3 / 254 (1 / 252)						
User Data Support == ON							

Table 2-7 PDU length value table

⁶ If User Data Support is OFF, the maximum allowed Pdu Length indicates the length without the bytes reserved for user data. The maximum allowed Pdu Length is always 254, but the user data bytes cannot be used in this case, and thus the given maximum Pdu Length is only a recommendation.

2.6.4 Decoupled Transmissions

FlexRay Network Management supports both buffer access methods, immediate and decoupled buffer access. Which method is used can be configured for each transmitted NM PDU independently (refer to [9]).

When immediate buffer access is chosen, the NM PDU data is copied immediately with the service call `FrIf_Transmit` (see also chapter 2.6.1 ‘AUTOSAR FlexRay Interface’) into the hardware buffer.

When decoupled buffer access is used the NM PDU data in the service call `FrIf_Transmit` is ignored. Instead shortly before the transmission of the NM message the function

`FrNm_TriggerTransmit()` (4.4.1.3)

is called by the FlexRay Interface and the NM PDU data is copied within this function by the Network Management.

Note that this service is not available for passive nodes since they do not transmit NM messages.

The usage of this feature is optional and can be configured (see [9]).

2.6.5 NM Hardware Vector Support

The Network Management Vector Support allows an automated aggregation of the NM PDU data in the static segment. Therefore the controller applies a bitwise OR to each received NM PDU data. The length of the NM Vector can be configured from 0 to 12 bytes and the data aggregated by this feature is called NM Vector.

The NM Vector is completely aggregated when all NM votes of one cycle are received and can be read out by the NM afterwards with the service call

`FrIf_GetNmVector()` (4.3)

The usage of this feature is optional and can be configured.

For information about this API refer to [9]

2.6.6 Dual Channel Support

FlexRay NM supports the sending of NM messages on both FlexRay channels. While in the static segment this is automatically supported by the FlexRay Interface, the FlexRay NM has to provide the support for the dynamic segment.

The following table shows the respective number of NM PDUs needed according to the Dual Channel and Schedule Variant parameter:

Schedule Variant	PDU Schedule Variant 1	PDU Schedule Variant 2	PDU Schedule Variant 3	PDU Schedule Variant 4	PDU Schedule Variant 5	PDU Schedule Variant 6	PDU Schedule Variant 7
Configuration Parameter							
Dual Channel Support = ON / Sync Pdu Master Enabled = OFF	1	2	2	3	3	4	3
Dual Channel Support = OFF / Sync Pdu Master Enabled = OFF	1	1	2	2	2	2	2
Dual Channel Support = ON / Sync Pdu Master Enabled = ON	N/A ⁷	4	N/A ⁷				

Table 2-8 Number of Tx-PDUs needed

Note that this service is not available for passive nodes since they do not transmit NM messages.

The usage of this feature is optional and can be configured.

2.6.7 User Data

The user data for the NM message transmitted next on the bus can be set by the service:

`FrNm_SetUserData()` (4.2.2.6.1)

Note that this service is not available for passive nodes since they do not transmit NM messages.

The service

`FrNm_GetUserData()` (4.2.2.6.2)

allows reading the user data of the last received message on the bus.

As the NM PDU layout is configurable, the user data placement depends on the given configuration. The PDU layout and the content of the user data are OEM specific and therefore provided by the OEM.

The usage of this feature is optional and can be configured.

Note that for setting of NM user data a second possibility can be configured. If the feature 'Com User Data Support' is used the API `FrNm_SetUserData()` is not available.

⁷ Currently not supported Schedule Variants

2.6.8 Voting Cycle and Data Cycle

The settings ‘Voting Cycle’ and ‘Data Cycle’ [unit: FlexRay Cycles] influence the periodicity of the NM PDU transmission requests. A setting of ‘Data Cycle’ being four leads to an NM Data PDU transmission request in every fourth FlexRay cycle. Likewise, a setting of ‘Voting Cycle’ being eight leads to an NM Vote PDU⁸ transmission request in every eighth FlexRay cycle. The moments where the NM PDU(s) can be seen on the bus, i.e. in which FlexRay cycle(s) the NM PDU(s) occur on the bus, depend on the Frame Triggerings of the Frames where the NM PDU(s) reside in.

If there is only one NM TX PDU per ECU (e.g. PDU Schedule Variant is 2 and Dual Channel Pdu Enabled is OFF – see also Table 2-8) and this NM PDU is only being sent in one Frame which has one Frame Triggering, the following constraint needs to be considered: the Data Cycle setting has to be equal to the ‘Cycle Repetition’ setting of the Frame Triggering. This requires that all nodes are able to send their NM PDU each n^{th} FlexRay cycle, $n := \text{‘Data Cycle’}$.

If there is more than one NM TX PDU per ECU (e.g. PDU Schedule Variant is 2 and Dual Channel Pdu Enabled is ON), the NM PDUs are most likely sent in at least two different frames. The Frame Triggerings of these frames need to have the same ‘Cycle Repetition’ setting.



Note

Typically the Data Cycle and Voting Cycle are specified by the OEM.

2.6.8.1 Examples

Let ‘Data Cycle’ be 8, PDU Schedule Variant be 2. In Figure 2-3 there are five nodes that send NM PDUs. The ‘Cycle Repetition’ of each Frame Triggering of every Frame where an NM PDU resides in is 8, too. The ‘Base Offset’ setting is different for each Frame Triggering so that the NM PDU can be sent by every node. The nodes are all able to send their NM PDUs, since each transmission request is repeated every eighth cycle and since the Frame Triggering settings have all a different Base Offset setting.

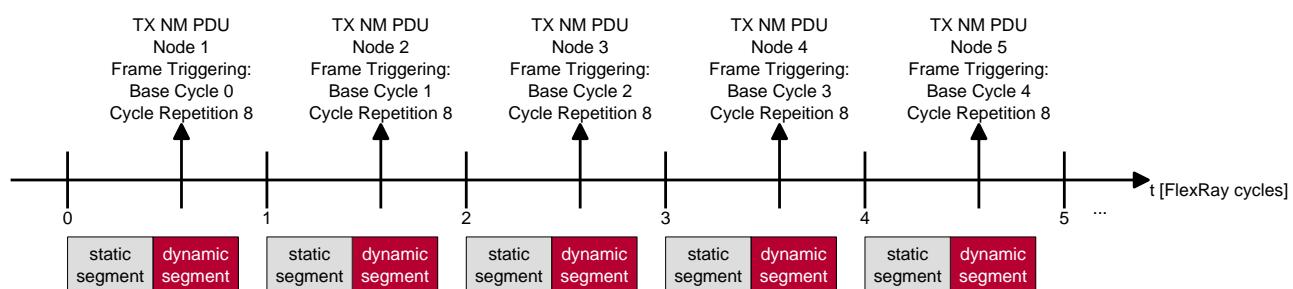


Figure 2-3 Example for Data Cycle = 8 with five nodes sending their NM PDUs

Let ‘Data Cycle’ be 4. In Figure 2-4 there are also five nodes that send NM PDUs. Node 5 has the same Frame Triggering settings as Node 4, so if all nodes need to send their NM

⁸ Separate NM Vote PDUs are only available in PDU Schedule Variants 3, 4, 5, 6, and 7. Otherwise, the NM Vote PDU and the NM Data PDU are the same object.

PDU, the NM PDU of Node 4 or Node 5 might not be sent. This depends on the Slot ID in the dynamic segment of the Frame Triggering, i.e. which frame comes first. Due to the length of the frame, other frames that have to be sent in a succeeding minislot, cannot be sent because the minislots are still blocked. Thus, this example is not allowed.

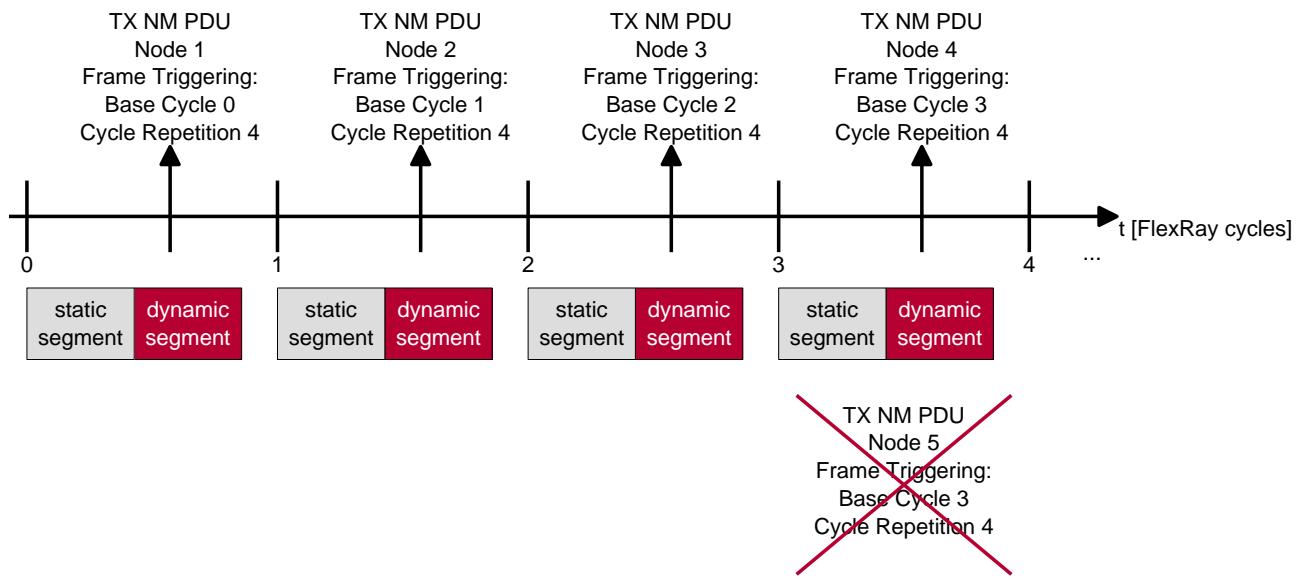


Figure 2-4 Example for Data Cycle = 4 with five nodes where one node eventually cannot send its NM PDU

For the next example, let 'Data Cycle' be 2 for Node 1, 'Data Cycle' be 4 for Node 2 and 3. Figure 2-5 illustrates three nodes sending NM PDUs. All nodes can send their NM PDUs. Node 1 transmits its NM PDU in the same frame in this example. It is also possible that Node 1 would send its NM PDU in two separate frames, both having a Frame Triggering with a Cycle Repetition of 2.

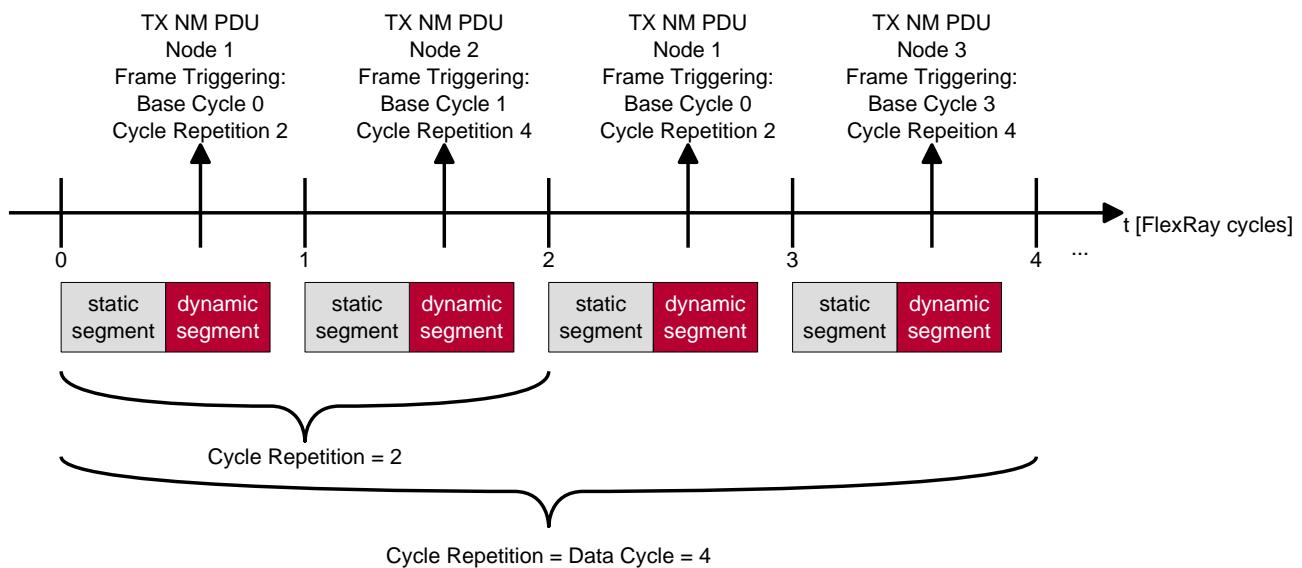


Figure 2-5 Example for Node 1 having 'Data Cycle' 2 and Nodes 2 and 3 having 'Data Cycle' 4

2.7 Node Detection

In order to detect which nodes are currently present within the network, this mechanism can be used in the network mode. If a network node requests node detection, the requesting

node performs a transition to Repeat Message State and sets the Repeat Message Bit within the Control Bit Vector of the NM PDU by calling the service

FrNm_RepeatMessageRequest() (4.2.2.7.1)

Upon reception of the Repeat Message Bit all network nodes perform a transition to Repeat Message State. This allows the requesting node to collect all source node identifiers from active nodes.

The local source node identifier can be retrieved by the service

FrNm_GetLocalNodeIdentifier() (4.2.2.7.3)

The source node identifier from the last received message can be retrieved by the service

FrNm_GetNodeIdentifier() (4.2.2.7.2)

The usage of this feature is optional and can be configured.

2.8 NM PDU Receive Indication

The NM Interface is notified about the reception of an NM message by the function

Nm_PduRxIndication() (4.3)

This callback is given directly to the NM Interface in context of the function FrNm_RxIndication (see chapter 4.4.1.2 ‘FrNm_RxIndication: NM PDU Reception Indication’).

The usage of this feature is optional and can be configured.

2.9 Gateway Functionality

2.9.1 Remote Sleep Indication and Cancellation

In order to synchronize networks it might be necessary to get an indication whether no more network nodes require bus-communication. This is the so-called ‘Remote Sleep Indication’. The start of the remote sleep indication is indicated by

Nm_RemoteSleepIndication() (4.3)

If any NM message is received during Normal Operation State after the remote sleep indication occurred the NM Interface is notified about ‘Remote Sleep Cancellation’ by calling

Nm_RemoteSleepCancellation() (4.3)

It is also possible to retrieve the current remote sleep state by calling the service:

FrNm_CheckRemoteSleepIndication() (4.2.2.8.1)

The remote sleep state can only be checked in the Network Mode.

The usage of this feature is optional and can be configured.

2.9.2 Bus Synchronization

The basis of the FlexRay bus depends on the fact that it is synchronous. Therefore the need to request synchronization is not given. This service call does not implement any functionality.

FrNm_RequestBusSynchronization() (4.2.2.9.1)

The usage of this feature is optional and can be configured.

2.10 Coordinator Synchronization Support

For supporting the NM Interface Coordinator Synchronization with more than one coordinator connected to the same channel it is necessary to provide an additional bit in the CBV. Therefore the service call

FrNm_SetSleepReadyBit() (4.2.2.12)

allows setting and clearing the NM Coordinator Sleep Ready Bit (see chapter 2.6.3 for further information).

2.11 Error Handling

2.11.1 Development Error Reporting

By default, development errors are reported to the DET using the service `Det_ReportError()` as specified in [4], if development error reporting is enabled (i.e. pre-compile parameter `FRNM_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 FRNM ID is 32.

The reported service IDs identify the services which are described in chapter 4.2 and 4.4. The following table presents the service IDs and the related services:

Service ID	Service
0x00	FrNm_Init
0x01	FrNm_PassiveStartUp
0x02	FrNm_NetworkRequest
0x03	FrNm_NetworkRelease
0x05	FrNm_EnableCommunication
0x06	FrNm_SetUserData
0x07	FrNm_GetUserData
0x08	FrNm_GetPduData
0x09	FrNm_RepeatMessageRequest
0x0A	FrNm_GetNodeIdentifier
0x0B	FrNm_GetLocalNodeIdentifier
0x0C	FrNm_DisableCommunication
0x0D	FrNm_CheckRemoteSleepIndication
0x0E	FrNm_GetState
0x0F	FrNm_GetVersionInfo
0x11	FrNm_Transmit
0x12	FrNm_SetSleepReadyBit
0xC0	FrNm_RequestBusSynchronization

Service ID	Service
0xE0	FrNm_TxConfirmation
0xE1	FrNm_RxIndication
0xE4	FrNm_TriggerTransmit
0xEF	FrNm_StartupError
0xF0 + <NmClstIdx>	FrNm_MainFunction_X
0xF3	FrNm_RequestSynchronizedPncShutdown

Table 2-9 Service IDs

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

Error Code	Description
0x01	FRNM_E_UNINIT API service used without module initialization.
0x02	FRNM_E_INVALID_CHANNEL API service used with wrong channel handle.
0x03	FRNM_E_INVALID_POINTER API service called with invalid parameter.
0x04	FRNM_E_PDU_ID_INVALID API service called with invalid PDU ID.
0x05	FRNM_E_ALREADY_INITIALIZED The service FrNm_Init() is called while the module is already initialized
0x06	FRNM_E_INVALID_GENDATA The value of a variable used for write accesses is out of bounds
0x10	FRNM_E_INIT_FAILED ⁹ FrNm initialization has failed.
0x11	FRNM_E_RXINDICATION_DLC_ERROR ⁹ PDU Length mismatch in Frf API with configured Length.
0x12	FRNM_E_PDUR_TRIGGER_TX_ERROR ⁹ Call of PduR_FrNmTriggerTransmit failed.
0x13	FRNM_E_INVALID_PN_SYNC_SHUTDOWN_REQUEST PNSR message is received on a channel that is actively coordinated.
0x14	FRNM_E_TRANSMISSION_OF_PN_SHUTDOWN_MESSAGE_FAILED PNSR message could not be transmitted in the configured time.
0x15	FRNM_E_REQUEST_PN_SHUTDOWN_MESSAGE_INVALID_STATE API service FrNm_RequestSynchronizedPncShutdown is called in an invalid NM state.

Table 2-10 Errors reported to DET

2.11.1.1 Parameter Checking

AUTOSAR requires that API functions check the validity of their parameters. The checks are internal parameter checks of the API functions. These checks are for development error reporting. The Parameter FRNM_DEV_ERROR_DETECT dis/enables the call of Det_ReportError() for all checks globally.

⁹ Error is a Vector extension

2.11.2 Production Code Error Reporting

By default, production code related errors are reported to the DEM using the service `Dem_ReportErrorStatus()` as specified in [5], if production error reporting is enabled.

If another module is used for production code error reporting, the function prototype for reporting the error can be configured by the integrator, but must have the same signature as the service `Dem_ReportErrorStatus()`.

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

Error Code	Description
N/A	Currently no DEM errors are specified

Table 2-11 Errors reported to DEM

2.12 Com User Data Support

The FlexRay NM supports the possibility to write the NM user data via Com signals. Therefore the signals have to be provided within an additional I-PDU in the configuration. The FlexRay NM updates its transmission buffer each time before sending a NM message with the current data. Therefore it calls the function:

`PduR_FrNmTriggerTransmit()` (4.3)

When the NM message has been successfully transmitted the confirmation is forwarded to the upper layer by calling the function:

`PduR_FrNmTxConfirmation()` (4.3)

Depending on the signal and I-PDU configuration a signal change can lead to a request for an immediate NM message transmission by calling the function

`FrNm_Transmit()` (4.2.2.10.1)

As FlexRay NM cannot send NM messages outside its configured slots, the function call will always return `E_OK` but will not execute any functionality.

The following chapters describe the configuration preconditions of this feature in more details.

Note that some additional configuration for this feature has to be done in the PDU router.

2.12.1 Configuration Preconditions for AUTOSAR ECU Configurations

For using this feature some additional configuration content within the AUTOSAR system description / ECU Extract is necessary. The following table provides an overview of the items that have to be added to the system description.

Configuration Element	Description
Signal I-PDU	For each NM message one signal I-PDU must be configured. An appropriate signal mapping to the I-Signals has to be defined here. I-PDUs are defined in the ECU-specific part.
I-Signal	Multiple system signals can be defined for each NM message. At least one signal is required. I-Signals are defined in the ECU-specific part and refer to a system signal.

Configuration Element	Description
System Signal	For each I-Signal a corresponding system signal is necessary which defines length, data type and init value.
I-PDU Port	For each I-PDU an I-PDU port with the communication direction 'OUT' is required.
Signal Port	For each signal a signal port with the communication direction 'OUT' is required.
I-PDU Triggering	For each I-PDU an I-PDU triggering is required that references to the corresponding I-PDU port and the signal I-PDU.
Signal Triggering	For each I-Signal a signal triggering is required that references to the corresponding signal port and I-Signal.

Table 2-12 Configuration Precondition Overview for AUTOSAR ECU Configurations

Additionally, a reference from the NM PDU to the related I-PDU with the signals must be established by adding 'ISignalToIPduMappings' to the NM PDU. The following example demonstrates how this should be done:

```
<NM-PDU>
  <SHORT-NAME>NM_PDU</SHORT-NAME>
  <LENGTH>8</LENGTH>
  <I-SIGNAL-TO-I-PDU-MAPPINGS>
    <I-SIGNAL-TO-I-PDU-MAPPING>
      <SHORT-NAME>NM_USR_DT </SHORT-NAME>
      <I-SIGNAL-REF DEST="I-SIGNAL"/>ISignal/NM_USR_SIGNAL</I-SIGNAL-REF>
      <PACKING-BYTE-ORDER>MOST-SIGNIFICANT-BYTE-LAST</PACKING-BYTE-ORDER>
      <START-POSITION>32</START-POSITION>
    </I-SIGNAL-TO-I-PDU-MAPPING>
  </I-SIGNAL-TO-I-PDU-MAPPINGS>
</NM-PDU>
```

2.13 Active Wake-up Bit

The mode change from Bus-Sleep Mode to Network Mode triggered by a network request is specified as 'Active Wake-up'. Upon an Active Wake-up the FlexRay NM sets the active wake-up bit within the Control Bit Vector at bit position 4.

This feature is optional and is activated if the Control Bit Vector Active parameter is ON.

2.14 Car Wake-up

Every ECU shall be able to wake up all other ECUs of the car. This wake-up request information is contained in the NM message user data of an ECU. The central gateway ECU evaluates the Car Wake-up request information and wakes up all connected communication channels.

This feature is optional and has to be configured.

2.14.1 Rx Path

If the FlexRay NM receives a NM message it evaluates the user data content. If the car wake-up bit is set and the Node ID passes a filter (if Node ID filter is enabled) the FlexRay NM notifies the NM via the following callback function:

`Nm_CarWakeUpIndication()` (4.3)

2.14.2 Tx Path

For the transmission of the car wake-up bit it has to be set at the corresponding location within the NM user data. It is then transmitted within the next NM message.

**Info**

It is recommended to use the feature ‘Com User Data Support’ for the transmission path.

2.15 Partial Networking

To reduce the power consumption of ECUs it shall be possible to switch off the communication stack during active bus communication. For FlexRay clusters it is not possible to switch off only one or some ECUs but the FlexRay ECUs may require information from network clusters on CAN. Therefore the FlexRay NM must be also able to evaluate the cluster request information (CRI) which is contained in the NM message user data. The FlexRay NM provides the aggregated PN information to the upper layer by updating the content of additional I-PDUs in the Com.

Algorithm details are described in the following sub-chapters.

This feature is optional and has to be configured.

**Note**

The highest PnInfoByte must be configured below the 64th byte of the NM message. This means there can be a maximum of 63 PnInfoBytes.

2.15.1 Availability of Partial Network Request Information

To distinguish between NM messages containing PN cluster request information (CRI) and NM messages without CRI a special bit in the control bit vector (bit 6) is used. Only if this bit is set the NM message contains PN information and will be processed by the algorithm.

2.15.2 Transmission of the CRI Bit

The FlexRay NM sets the CRI bit at bit position 6 in the Control Bit Vector to 1 for each channel if the Partial Networking feature is enabled on the corresponding channel.

2.15.3 Extended Handling for Received NM Messages

If the CRI bit is cleared the FlexRay NM performs the standard NM message reception handling.

If the CRI bit is set the FlexRay NM evaluates the CRI content of the NM message. The location and the length of the CRI in the NM user data can be configured. Each bit within the CRI content represents one cluster. The corresponding cluster is being requested if and only if the bit that belongs to the cluster is set. Because not every cluster is relevant for the ECU a configurable PN filter mask is applied to the CRI content. Irrelevant cluster requests can be ignored by setting the corresponding bit in the filter to 0. If at least one bit within the received PN information matches with a bit in the PN filter mask the NM message is relevant for the ECU, otherwise the NM message is not relevant for the ECU.

If a NM message is not relevant the standard NM message reception handling is done.

If a NM message is relevant the FlexRay NM performs the standard NM message reception and additionally the filtered PN content of this message is used for the further PN algorithm.

2.15.4 Aggregation of Requested Partial Networks

The FlexRay NM saves all relevant cluster requests in an array. This requested clusters array is updated with every relevant NM message.

The FlexRay NM aggregates requested PN information by two slightly different algorithms. First the external (received) and internal (sent) PN requests are aggregated over all networks (channels) to a combined state called External Internal Requests Aggregated (EIRA). Second only the external (received) PN requests are aggregated for each network to the so called External Requests Aggregated (ERA) state. Both algorithms can be activated independently in the configuration.

For the EIRA algorithm every received or sent NM message on any network is evaluated and the relevant PN information (according to the PN filter mask and the CRI bit) is combined to one aggregated state. Therefore this state contains the information which partial networks are active on the whole ECU.

The ERA algorithm performs the evaluation of the received NM messages and storage of the relevant PN information (according to the PN filter mask and the CRI bit) per network. Therefore the ERA state contains for each network the information which partial networks are requested by other ECUs and have to be active due to external needs.

Whenever a cluster is requested the first time (i.e. a bit is set the first time within this PN information) the new request is stored and a timer is started. When the request is repeated before the timer elapses the timer is restarted. When the timer elapses the request is deleted.

Any change (storing or deleting a request) within the EIRA or ERA leads to an update of the content of the EIRA or ERA I-PDU in the Com. Therefore the following function is called with the corresponding EIRA or ERA PDU handle:

`PduR_FrNmRxIndication()` (4.3)

Note that one ERA I-PDU exists for each network.

2.15.5 Spontaneous Sending of NM Messages

As FlexRay NM cannot send NM messages outside its configured slots it does not support spontaneous sending of NM messages.

2.15.6 Transmission of CRI bit without Partial Networking

It is possible to transmit an NM message with only the CRI bit set in the CBV, but Partial Networking feature disabled. This is done by enabling the switch ‘Cri Bit Always Enabled’ in the DaVinci Configurator.

2.15.7 Synchronized PNC Shutdown

The synchronized PNC shutdown enables the possibility to shutdown a PNC synchronously in the entire PNC topology.

If this feature is not enabled, it can occur, that a channel of an ECU shuts down its PNC earlier than other ECUs. This can lead to timeouts in the application.

If the Top-Level PNC Coordinator decides to release the PNC a shutdown message will be transmitted and forwarded by the Intermediate PNC Coordinators until it reaches all the leaf nodes. On reception of this message the PnResetTimer is reset, which leads to the synchronized PNC shutdown over all the ECUs.

For more information please refer to [3].

2.15.7.1 Synchronized PNC Shutdown Restrictions

The MICROSAR Classic FrNm does not support reliable TxConfirmation. Therefore the successful transmission is monitored via a MsgTimeoutTimer that is set, when a message is requested to be transmitted.

This can lead to scenarios, where the regular NM message is tried to be sent, but is not yet confirmed via a TxConfirmation. Then the PNSR message is transmitted and subsequently a TxConfirmation is received for the previously sent regular NM message. The FrNm state machine accepts this TxConfirmation for the PNSR message and will not retry to send it again, in case the PNSR could actually not be transmitted.



Note

This is a very rare scenario and should not occur in the common use cases.

If there was no TxConfirmation, this would usually lead to a Sync Loss and no further Nm messages are sent.

2.16 Sync PDU Master in Dual Channel Use Case

In the Dual Channel Use Case (channels A+B are used by FlexRay cluster), FrNm can be used on both physical channels to coordinate the shutdown of networks. If the NM PDUs are sent within frames in the dynamic segment, this can be realized through the ‘Dual Channel Pdu Enabled’ setting.

However, if there are also ECUs that are only connected to one of the physical FlexRay channels (i.e. either channel A or channel B), somehow the request of those ECUs to keep the network awake needs to be forwarded to the other physical channel.

For this purpose, FrNm has been extended by the ‘Sync PDU Master’ functionality. There is one node in the network that is configured as ‘Sync PDU Master’. This node sends so called additional NM Sync PDUs, one on each physical channel.

**Note**

This feature requires ‘Passive Mode Enabled’ to be turned OFF, ‘Dual Channel Pdu Enabled’ to be turned ON and ‘Pdu Schedule Variant’ to be set to ‘V2’.

To enable the feature, use the ‘Sync Pdu Master Enabled’ setting. Configure the FlexRay cycle offset where the transmit request of the NM Sync PDUs shall be issued from FrNm_MainFunction_X in the ‘Sync Pdu Tx Request Cycle Offset’ setting. This setting is related to the Voting Cycle setting. Make sure that the ‘Tx Pdu Is Sync Pdu’ setting is turned ON for the two NM Sync PDUs.

2.16.1 Example Scenario

Let ECU 1 be equipped with FrNm on FlexRay channel A and B.

Let ECU 2 be equipped with FrNm on FlexRay channel A, let it be also connected to channel B without FrNm.

Let ECU 3 be equipped with FrNm on FlexRay channel B, let it be also connected to channel A without FrNm.

Let ECU 4 be equipped with FrNm on FlexRay channel A, let it not be connected to channel B.

Let ECU 5 be equipped with FrNm on FlexRay channel B, let it not be connected to channel A.

This setup is summarized in Figure 2-6.

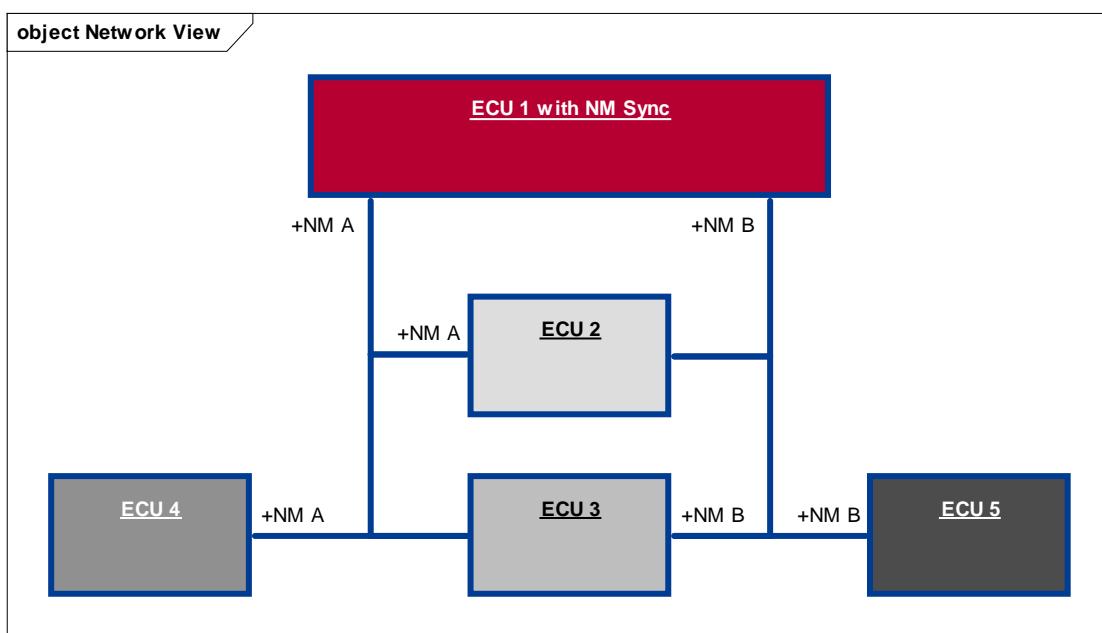


Figure 2-6 Example scenario for Sync PDU Master usage

ECU 1 shall synchronize requests to keep the network awake between the FlexRay channels A and B.

This is required, because if ECU 2 or ECU 4 need to keep the network awake, this information needs also to be shared with ECU 3 and ECU 5, which are not connected to FlexRay channel B.

This is also required, because if ECU 3 or ECU 5 need to keep the network awake, this information needs to be shared with ECU 2 and ECU 4.

Thus ECU 1 somehow needs to forward the requests from the ECUs on one physical channel to the ECUs on the other physical channel. The way how this forwarding mechanism works, is explained as follows.

2.16.2 Sync PDU Master Algorithm

There can only be one Sync PDU Master in one FlexRay cluster. This node is connected to both physical channels and sends NM Sync PDUs in addition to its regular NM PDUs.

If 'Dual Chanell Pdu Enabled' is ON and 'Pdu Schedule Variant' is 'V2', the regular NM PDUs are sent in frames inside the dynamic segment. There is one regular NM PDU for each physical channel. In addition to that, there is one NM Sync PDU for each physical channel, so the 'Sync PDU Master' node sends two regular NM PDUs and two NM Sync PDUs. The other nodes receive these NM PDUs and there is no difference for those nodes between the regular NM PDUs and the NM Sync PDUs.

As follows, the algorithm and further details of the Sync PDU feature is described from the Sync PDU Master point of view.

The algorithm of the NM Sync PDU Master node can be formulated like this:

During initialization of FrNm:

- > Clear the flag to send NM Sync PDUs.

If the FrNm state is not Bus Sleep:

- > At the beginning of a Repetition Cycle: clear the flag to send NM Sync PDUs.
- > At the beginning of a Data Cycle: if the regular NM PDUs are sent, set the flag to send NM Sync PDUs.
- > On reception of an NM PDU: set the flag to send NM Sync PDUs.
- > During the Repetition Cycle in a cycle that matches the 'Sync Pdu Tx Request Cycle Offset': if the flag to send NM Sync PDUs is set, issue the send request of the NM Sync PDUs.

This algorithm is depicted in Figure 2-7. The Idle states refer to activity outside FrNm. The algorithm is realized in the context of the FrNm_MainFunction_X functions (chapter 4.2.1.3) and FrNm_RxIndication (chapter 4.4.1.2). The beginning of a Repetition Cycle is reached if the modulo operation of the current cycle divided by Repetition Cycle equals zero (cycle % Repetition Cycle == 0). The beginning of a Data Cycle is reached if the modulo operation of the current cycle divided by Data Cycle equals zero (cycle % Data Cycle == 0. 'Network Released' refers to the information that FrNm_NetworkRelease (chapter 4.2.2.5) has been recently called or FrNm_NetworkRequest (chapter 4.2.2.4) has never been called since the

initialization of FrNm. ‘Decision Cycle’ refers to the setting ‘Sync Pdu Tx Request Cycle Offset’.



Note

Note that the feature requires ‘Pdu Schedule Variant’ to be V2. This setting requires ‘Data Cycle’ = ‘Voting Cycle’.

‘Repetition Cycle’ needs to be an integer multiple of ‘Voting Cycle’.

Therefore, ‘cycle % Repetition Cycle == 0’ implies ‘cycle % Data Cycle == 0’.

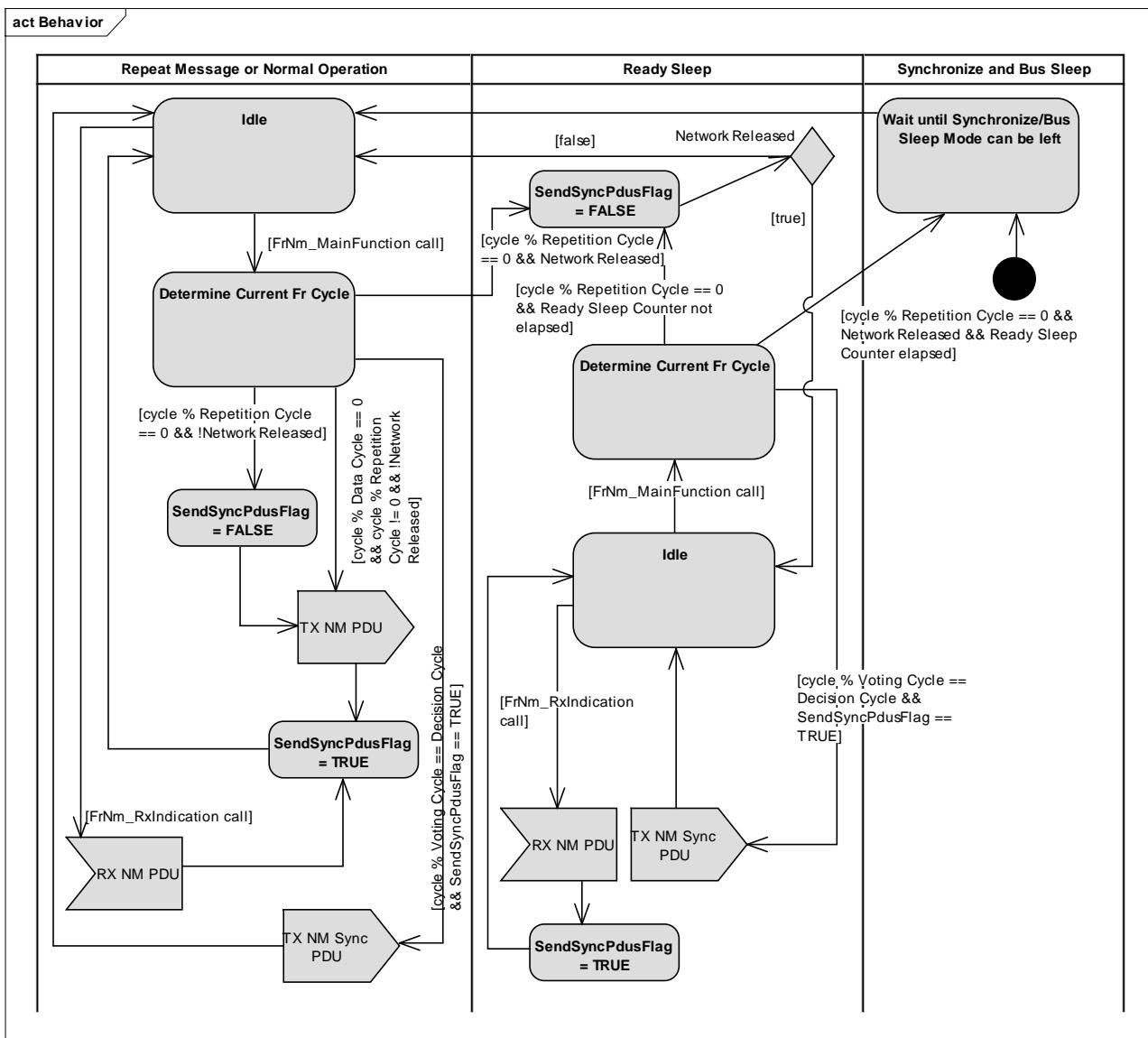
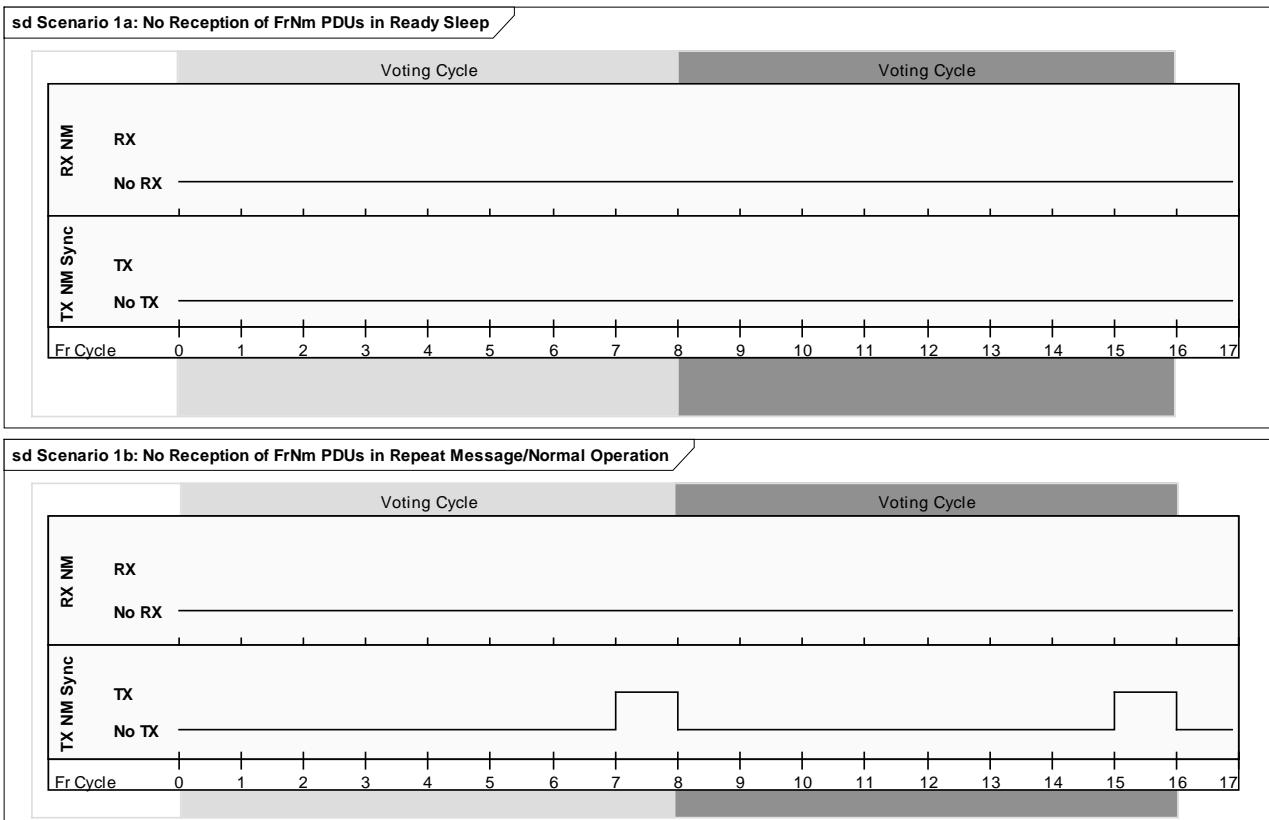


Figure 2-7 NM Sync PDU Master Algorithm

The effects of the algorithm are that the ‘Sync PDU Master’ node sends the NM Sync PDUs if and only if the node itself wants to keep the network awake (node is in Repeat Message

or Normal Operation) and/or if another node wants to keep the network awake (node is in Repeat Message, Normal Operation or Ready Sleep). If it does not want to keep the network awake but other nodes do, the NM Sync PDUs are sent for the rest of the Repetition Cycle after the reception of an NM PDU.

In the following example, the effects are depicted in Figure 2-8. For this example, Voting Cycle := 8 [FlexRay cycles], Data Cycle := 8 [FlexRay cycles], Repetition Cycle := 16 [FlexRay cycles], ‘Sync Pdu Tx Request Cycle Offset’ := 6 [Offset in each Voting Cycle], ‘Main Across Fr Cycle’ = FALSE.



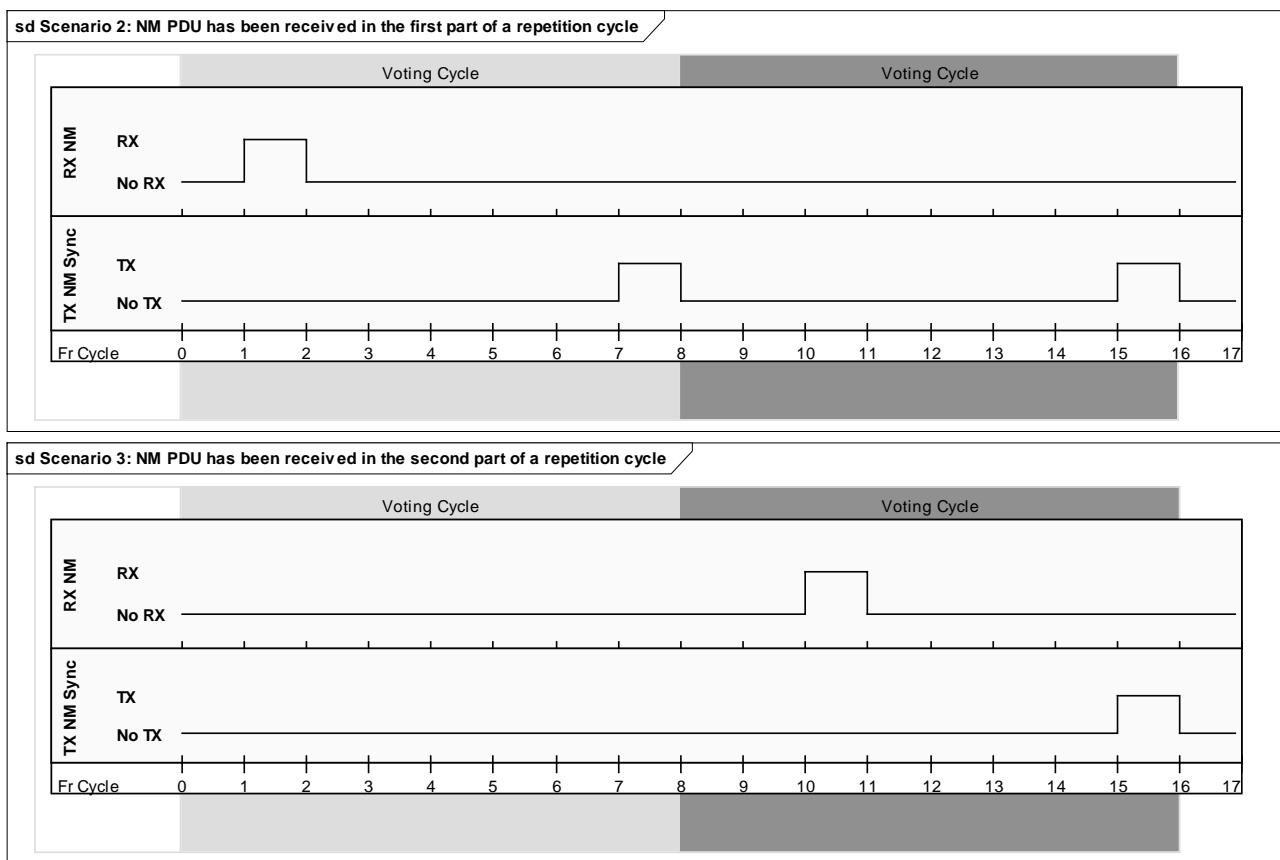


Figure 2-8 Different scenarios for Sync PDU Master

If FrNm is in Ready Sleep and no NM PDUs are received, no NM Sync PDUs are transmitted (Scenario 1a). If FrNm is in Repeat Message or Normal Operation, NM PDUs and NM Sync PDUs are transmitted (Scenario 1b). If FrNm is in Repeat Message, Normal Operation or Ready Sleep and NM PDUs are received in the first Voting Cycle of the Repetition Cycle, NM Sync PDUs are sent in both Voting Cycles. If FrNm is in Ready Sleep and NM PDUs are only received in the second Voting Cycle of the Repetition Cycle, NM Sync PDUs are only sent in the second Voting Cycle of the Repetition Cycle.



Note

Figure 2-8 does not precisely indicate in which slot the PDUs are sent/received, only in which cycle.

The transmit request of an NM Sync PDU is issued in the sixth cycle of a Voting Cycle since 'Main Across Fr Cycle' is FALSE and the FrNm_MainFunction_X needs to be executed at the end of the FlexRay cycle.

The frame triggering cycle offset of the frame where the NM Sync PDU is located is 7 in this example.

The NM Sync PDUs have the same length as regular NM PDUs. Their content is 0x00 for every data byte.

3 Integration

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

3.1 Scope of Delivery

The delivery of the FRNM contains the files which are described in the chapters 3.1.1 and 3.1.2:

3.1.1 Static Files

File Name	Source Code Delivery	Object Code Delivery	Description
FrNm.c	■		Source code of the FlexRay NM. The user must not change this file!
FrNm.h	■		API of FlexRay NM. The user must not change this file!
FrNm_Cbk.h	■		API of FlexRay NM callback functions. The user must not change this file!

Table 3-1 Static files



Do not edit manually

The static files listed above must not be edited by the user!

3.1.2 Dynamic Files

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

File Name	Description
FrNm_Cfg.c	Pre-compile variant configuration source file. The user must not change this file!
FrNm_Cfg.h	Configuration header file for all variants. The user must not change this file!
FrNm_Lcfg.c	Link-time variant configuration source file. The user must not change this file!
FrNm_PBcfg.c	Post-build variant configuration source file. The user must not change this file!

Table 3-2 Generated files

3.2 Include Structure

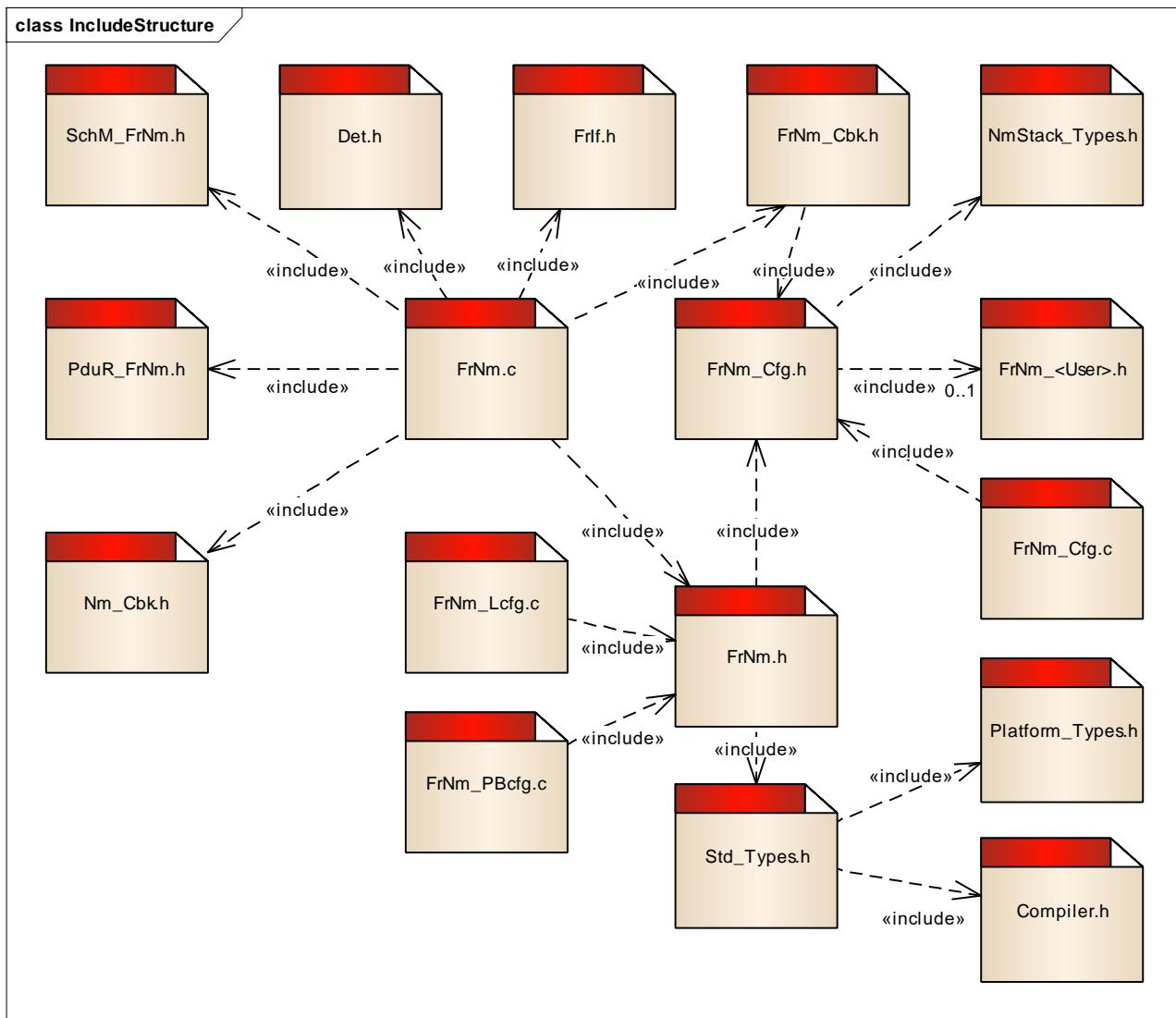


Figure 3-1 Include structure

3.3 Version Changes

Changes and the release versions of the FlexRay NM are listed at the beginning of the header and source code.

3.4 Main Function

The FlexRay main function has to be synchronized with the FlexRay cycle because almost all state transitions are bound to the beginning of a new cycle (refer to chapter 2.4 'Operational Modes and States'). Therefore the main function has to be called within a specific time slot, which is shown in the following figure:

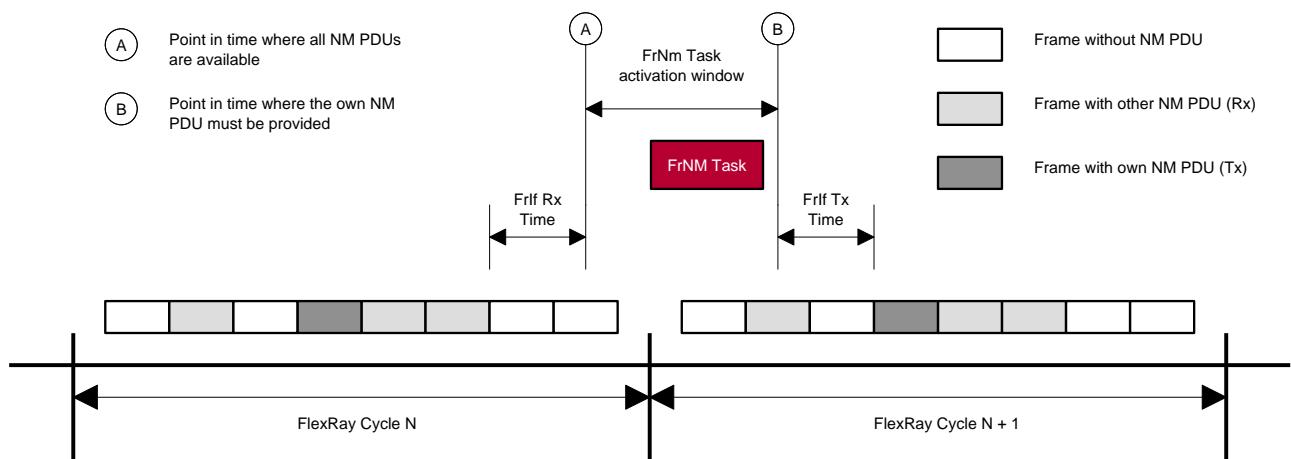


Figure 3-2 FlexRay NM Task Execution

The left border of this window is given by the point in time where at least all NM Votes are available to the NM. If Node Detection is enabled additionally all Control Bit Vectors must be available to the NM due to the impact of the Repeat Message Bit on the state transitions. The right border is defined by the point in time where the own NM PDU must be provided to the FlexRay Interface.

To ensure that the main function time slot is wide enough to allow a complete execution of the NM task some restrictions have to be applied that are not mentioned in [2]. They are discussed in chapter 2.1.3.3 ‘FlexRay NM Main Function Handling’.

It has to be configured if the NM task is called always at the end (‘Main Across Fr Cycle’ := FALSE) or the beginning of a FlexRay cycle (‘Main Across Fr Cycle’ := TRUE). The ‘Main Across Fr Cycle’ setting can be configured in the configuration tool. Figure 3-2 requires the ‘Main Across Fr Cycle’ to be TRUE.

The FlexRay NM main function has to be called on task level on every cycle. The main functions have to be called individually for each channel of the component. For further information refer to chapter 4.2.1.3 ‘FrNm_MainFunction: Channel-Specific Main Functions of FlexRay NM’.



Caution

The FlexRay NM main function must not be interrupted by any other task as this would influence the duration of the main function in an unpredictable way.



Note

If the BSW Scheduler is used, the main function calls are executed by the BSW Scheduler. When the BSW Scheduler is available in DaVinci Configurator the main function timings are automatically adapted when loading the configuration. Refer to [7] for more details about the BSW Scheduler.

**Note**

The FrNm main function needs to be called cyclically regardless whether the FlexRay bus is synchronized or not (with the configured 'Main Function Period').

3.4.1 OS Usage

If an OS is used, the task container where the FrNm main function is called, referred as FrNm task in the following, can for instance be synchronized against the FlexRay bus in the following ways:

1. Task Activation via synchronized Schedule Tables

Activate the FrNm task via Schedule Tables. These Schedule Tables shall be synchronized via the SyncScheduleTable() API.

Pro: precise handling, also on bus synchronization loss
Con: requires Scalability Class SC2 or SC4 of the OS

2. Task Activation through Relative Alarms

Call SetRelAlarm() multiple times for each task container (the first one of them being the FrNm task) on Cycle Start Interrupt.

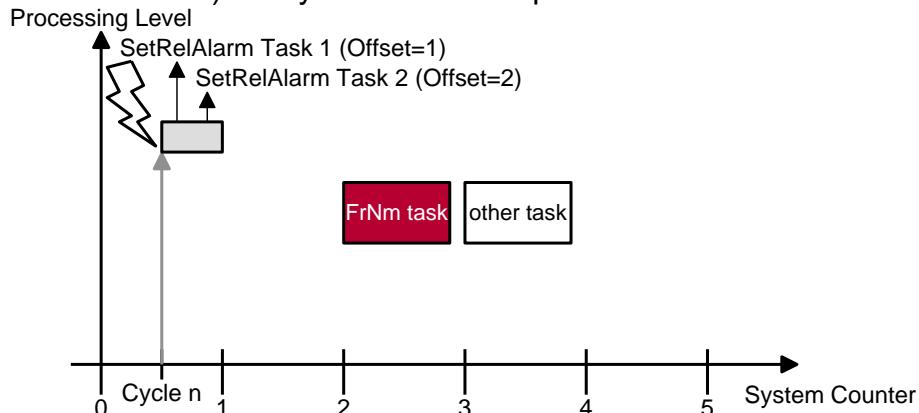


Figure 3-3 Task Activation through Relative Alarms

Pro: easy handling

Con: no critical sections are allowed according to OSEK for SetRelAlarm() calls.

Missing critical section may lead to an interruption between the SetRelAlarm() calls which may lead to a shift of the task containers resulting in a non-deterministic behavior. So the FrNm task has to be the first one that is scheduled by the SetRelAlarm() call.

If the FrNm task was not the first task scheduled by SetRelAlarm(), the effect would be as in Figure 3-4. Note that the same effect applies to the other tasks that are scheduled by the second and succeeding call of SetRelAlarm(), e.g. if the scheduling is like in Figure 3-3.

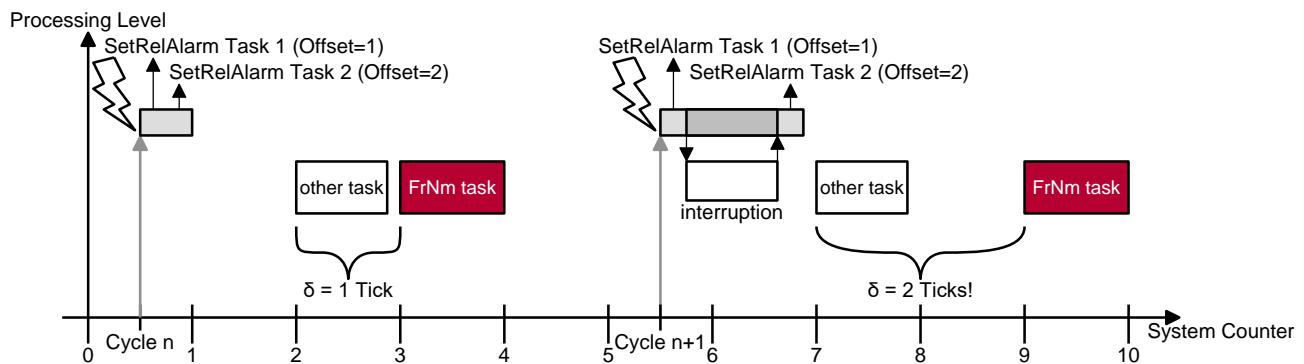


Figure 3-4 Task Activation through Relative Alarms if FrNm task is not the first task scheduled by SetRelAlarm

3. Task Activation in Job List Context

Call ActivateTask() for the FrNm task in the context where the FrIf Rx Job list with the last NM RX PDUs has been executed (see Figure 3-5).

Pro: easy handling

Con: no call if the FlexRay bus is not synchronized

Due to the missing execution of the Job List if the FlexRay bus is not synchronized, another task needs to be scheduled (e.g. by SetRelAlarm()) where the FrNm main function is called if the bus is not synchronized (the POC State determined by FrIf_GetPOCStatus() is not FR_POCSTATE_NORMAL_ACTIVE) as fallback solution.

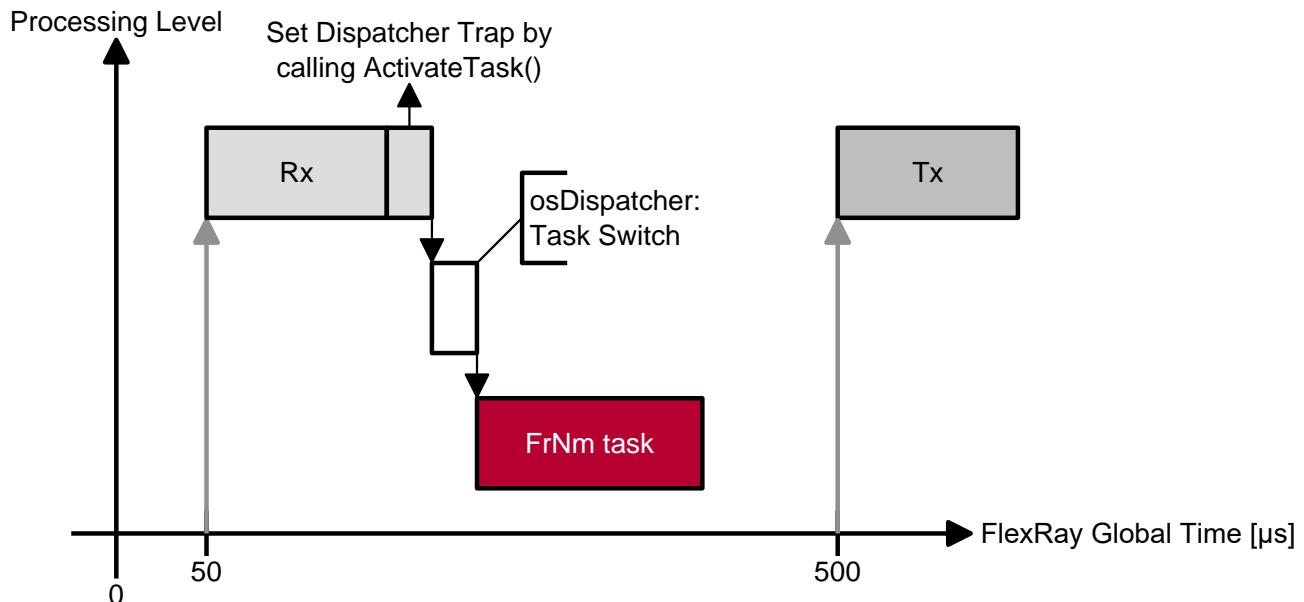


Figure 3-5 Task Activation in Job List Context



Note

This list is not complete.

3.5 Critical Sections

Critical Sections are handled by the RTE [7]. They are automatically configured by the DaVinci Configurator. User interaction is only necessary by updating the internal behavior using the solving action in DaVinci Configurator. It is signaled as a Warning in the Validation tab.

The FlexRay NM calls the following function when entering a critical section:

`SchM_Enter_FrNm_FRNM_EXCLUSIVE_AREA_i()` (6.6)

When the critical section is left the following function is called by the FlexRay NM:

`SchM_Exit_FrNm_FRNM_EXCLUSIVE_AREA_i()` (6.6)

Details which section needs what kind of interrupt lock are provided in chapter 3.6 ‘Critical Section Codes’.

3.6 Critical Section Codes

The FlexRay NM provides several critical section codes which must lead to corresponding interrupt locks, described in the following table:

Critical Section Define	Interrupt Lock
FRNM_EXCLUSIVE_AREA_0	No interruption by any interrupt is allowed. Therefore this section must always lock global interrupts.
FRNM_EXCLUSIVE_AREA_1	No interrupt locks are necessary if one of the following points can be ensured: <ul style="list-style-type: none"> A) The following two conditions are always true: <ol style="list-style-type: none"> 1. User data is not set via <code>FrNm_SetUserData()</code> API (e.g. when using feature ‘User Data via Com Enabled’) or <code>FrNm_SetUserData()</code> API cannot interrupt <code>FrNm_MainFunction()</code> API. 2. When using the Nm feature ‘Coordinator Synchronization Support’ the API <code>Nm_MainFunction()</code> cannot interrupt <code>FrNm_MainFunction()</code> API. B) All NM transmission messages are configured ‘decoupled’ in the FlexRay Interface (Parameter ‘FrflImmediate’ is disabled). Otherwise global interrupt locks are necessary.
FRNM_EXCLUSIVE_AREA_2	No interrupt locks are necessary if one of the following points can be ensured:

Critical Section Define	Interrupt Lock
	<p>A) The following three conditions are always true:</p> <ol style="list-style-type: none"> 1. User data is not set via <code>FrNm_SetUserData()</code> API (e.g. when using feature 'User Data via Com Enabled') or <code>FrNm_SetUserData()</code> API cannot interrupt <code>FrNm_TriggerTransmit()</code> API. 2. The API <code>FrNm_MainFunction()</code> cannot interrupt <code>FrNm_TriggerTransmit()</code> API. 3. When using the Nm feature 'Coordinator Synchronization Support' the API <code>Nm_MainFunction()</code> cannot interrupt <code>FrNm_TriggerTransmit()</code> API. <p>B) All NM transmission messages are configured 'immediate' in the FlexRay Interface (Parameter 'FrflImmediate' is enabled).</p> <p>Otherwise global interrupt locks are necessary.</p>
<code>FRNM_EXCLUSIVE_AREA_3</code>	No interrupt locks are necessary if the API <code>FrNm_SetUserData()</code> cannot be interrupted by the APIs <code>FrNm_MainFunction()</code> and <code>FrNm_TriggerTransmit()</code> . Otherwise global interrupt locks are necessary.
<code>FRNM_EXCLUSIVE_AREA_4</code>	No interrupt locks are necessary if the API <code>FrNm_RxIndication()</code> cannot be interrupted by <code>FrNm_MainFunction()</code> API. Otherwise global interrupt locks are necessary.
<code>FRNM_EXCLUSIVE_AREA_5</code>	No interrupt locks are necessary if the API <code>FrNm_RxIndication()</code> cannot be interrupted by the APIs <code>FrNm_GetUserData()</code> and <code>FrNm_GetPduData()</code> . Otherwise global interrupt locks are necessary.
<code>FRNM_EXCLUSIVE_AREA_6</code>	No interrupt locks are necessary if the API <code>FrNm_RxIndication()</code> cannot interrupt the APIs <code>FrNm_GetUserData()</code> and <code>FrNm_GetPduData()</code> . Otherwise global interrupt locks are necessary.
<code>FRNM_EXCLUSIVE_AREA_7</code>	No interrupt locks are necessary if the API <code>FrNm_RequestSynchronizedPncShutdown()</code> cannot interrupt the API <code>FrNm_MainFunction</code> . Otherwise global interrupt locks are necessary.

Table 3-3 Critical Section Codes

4 API Description

For an interfaces overview please see Figure 1-2.

4.1 Data Types

The software module FlexRay NM uses the standard AUTOSAR data types that are defined within `Std_Types.h`, the platform specific data types that are defined within `Platform_Types.h` and the Communication Stack Types defined within `ComStack_Types.h`. Furthermore the standard AUTOSAR NM Stack Types defined within `NmStack_Types.h` are used.

4.2 Services Provided by FRNM

4.2.1 Administrative Functions

4.2.1.1 FrNm_Init: Initialization of FlexRay NM

Prototype	
<pre>void FrNm_Init (const FrNm_ConfigType * const nmConfigPtr)</pre>	
Parameter	
nmConfigPtr	Configuration Pointer
Return code	
-	-
Functional Description	
Initialization of FlexRay NM and of its state machine By default the NM starts in the Bus-Sleep Mode and the bus-communication is released.	
Particularities and Limitations	
<ul style="list-style-type: none">> Service ID: see table 'Service IDs'> The function <code>FrNm_Init()</code> has to be called before any other FlexRay NM service function is called (except <code>FrNm_InitMemory()</code>).> This function is non-reentrant.> This function is synchronous.> Note that the syntax of the function prototype deviates from the AUTOSAR definition in [2].	
Expected Caller Context	
<ul style="list-style-type: none">> Task level	

Table 4-1 FrNm_Init

4.2.1.2 FrNm_InitMemory

Prototype

```
void FrNm_InitMemory (void)
```

Parameter

-	-
---	---

Return code

-	-
---	---

Functional Description

Initialize Memory with expected start values. Called on System Startup

Particularities and Limitations

- > Service ID: see table 'Service IDs'
- > Interrupts have to be disabled before calling this function.
- > This function is called from Application.
- > This function is reentrant.
- > This function is synchronous.

Call Context

- > Task level only during system initialization

Table 4-2 FrNm_InitMemory

4.2.1.3 FrNm_MainFunction: Channel-Specific Main Functions of FlexRay NM

Prototype

```
void FrNm_MainFunction_X ( void )
```

(x=0...Number of system channels)

Parameter

-	-
---	---

Return code

-	-
---	---

Functional Description

Main functions of FlexRay NM.

For each instance the corresponding main function has to be called with the system channel handle that belongs to this instance as postfix.

Particularities and Limitations

- > Service ID: see table 'Service IDs'
- > These functions have to be called cyclically on task level and synchronously to the FlexRay cycle.
- > These functions also have to be called cyclically while the FlexRay bus is not synchronized.
- > These functions are non-reentrant.
- > These functions are synchronous.
- > If these services are already called by the BSW Scheduler, they must not be called by the application.
- > Asynchronous task activation shall not be used to call FrNm_MainFunction_X according to [2]

Call context

- > Task level

Table 4-3 FrNm_MainFunction

4.2.2 Service Functions

4.2.2.1 FrNm_GetVersionInfo: Version Information API

Prototype

```
void FrNm_GetVersionInfo( Std_VersionInfoType* NmVerInfoPtr )
```

Parameter

NmVerInfoPtr	Pointer where the Version Information shall be copied to
--------------	--

Return code

-	-
---	---

Functional Description

FrNm_GetVersionInfo() returns version information, vendor ID and AUTOSAR module ID of the component. The versions are BCD-coded.

Particularities and Limitations

- > Service ID: see table 'Service IDs'
- > This function is reentrant.
- > This function is synchronous.
- > This function is available if FRNM_VERSION_INFO_API is STD_ON.

Call context

- > Task level

Table 4-4 FrNm_GetVersionInfo

**Caution**

Since there is no interface from FlexRay NM to any higher software layer except DET or NM Interface all the following API services must not be called by ComM or the application!

4.2.2.2 FrNm_GetState: Get the State of the Network Management

Prototype

```
Std_ReturnType FrNm_GetState (
    const NetworkHandleType NetworkHandle,
    Nm_StateType* const nmStatePtr,
    Nm_ModeType* const nmModePtr )
```

Parameter

NetworkHandle	Identification of the system channel
nmStatePtr	Pointer where state of the network management shall be copied to
nmModePtr	Pointer where the mode of the network management shall be copied to

Return code

E_OK	No errors
E_NOT_OK	Retrieval of NM state has failed

Functional Description

Get the state and the mode of the network management.

Particularities and Limitations

- > Service ID: see table 'Service IDs'
- > This function is reentrant.
- > This function is synchronous.
- > This function is called by NM Interface.

Call Context

- > Task and interrupt level

Table 4-5 FrNm_GetState

4.2.2.3 FrNm_PassiveStartUp: Network Management Wake-Up

Prototype	
Std_ReturnType FrNm_PassiveStartUp (const NetworkHandleType NetworkHandle)	
Parameter	
NetworkHandle	Identification of the system channel
Return code	
E_OK	No errors
E_NOT_OK	Start of the Network Management has failed
Functional Description	
Passive start-up of the FlexRay NM state machine. It triggers the transition from the Bus Sleep Mode to the Network Mode (Repeat Message State; via Synchronize Mode). This service has no effect if the current state is not Bus-Sleep Mode. In that case E_NOT_OK is returned.	
Particularities and Limitations	
<ul style="list-style-type: none">> Service ID: see table 'Service IDs'> This function is non-reentrant.> This function is asynchronous.> This function is called by NM Interface.	
Call Context	
<ul style="list-style-type: none">> Task and interrupt level	

Table 4-6 FrNm_PassiveStartUp

4.2.2.4 FrNm_NetworkRequest

Prototype	
Std_ReturnType FrNm_NetworkRequest (const NetworkHandleType NetworkHandle)	
Parameter	
NetworkHandle	Identification of the system channel
Return code	
E_OK	No errors
E_NOT_OK	Requesting the network has failed
Functional Description	
Request the network if the ECU needs to communicate on the bus.	
Particularities and Limitations	
<ul style="list-style-type: none">> Service ID: see table 'Service IDs'> This function is non-reentrant.> This function is asynchronous.> This function is called by NM Interface.> Not available for passive nodes (<code>FRNM_PASSIVE_NODE_ENABLED</code> is <code>STD_ON</code>).	
Call Context	
<ul style="list-style-type: none">> Task and interrupt level	

Table 4-7 FrNm_NetworkRequest

4.2.2.5 FrNm_NetworkRelease

Prototype	
Std_ReturnType FrNm_NetworkRelease (const NetworkHandleType NetworkHandle)	
Parameter	
NetworkHandle	Identification of the system channel
Return code	
E_OK	No errors
E_NOT_OK	Releasing the network has failed
Functional Description	
Release the network if the ECU doesn't have to communicate on the bus.	
Particularities and Limitations	
<ul style="list-style-type: none">> Service ID: see table 'Service IDs'> This function is non-reentrant.> This function is asynchronous.> This function is called by NM Interface.> Not available for passive nodes (<code>FRNM_PASSIVE_NODE_ENABLED</code> is <code>STD_ON</code>).	
Call Context	
<ul style="list-style-type: none">> Task and interrupt level	

Table 4-8 FrNm_NetworkRelease

4.2.2.6 User Data Handling

4.2.2.6.1 FrNm_SetUserData: Set User Data

Prototype	
<pre>Std_ReturnType FrNm_SetUserData (const NetworkHandleType NetworkHandle, const uint8 *nmUserDataPtr)</pre>	
Parameter	
NetworkHandle	Identification of the system channel
nmUserDataPtr	Pointer to the user data for the next transmitted NM message
Return code	
E_OK	No errors
E_NOT_OK	Setting of user data has failed
Functional Description	
Set user data for NM messages transmitted next on the bus	
Particularities and Limitations	
<ul style="list-style-type: none">> Service ID: see table 'Service IDs'> This function is non-reentrant.> This function is synchronous.> This function is called by NM Interface.> Not available for passive nodes (FRNM_PASSIVE_NODE_ENABLED is STD_ON).> Not available if Com User Data Support is ON (FRNM_COM_USER_DATA_SUPPORT is STD_ON).> Not available if User Data Enabled is OFF (FRNM_USER_DATA_ENABLED is STD_OFF).	
Call Context	
<ul style="list-style-type: none">> Task and interrupt level	

Table 4-9 FrNm_SetUserData

4.2.2.6.2 FrNm_GetUserData: Get User Data

Prototype	
<pre>Std_ReturnType FrNm_GetUserData (</pre>	
const NetworkHandleType NetworkHandle,	
const uint8 *nmUserDataPtr)	
Parameter	
NetworkHandle	Identification of the system channel
nmUserDataPtr	Pointer where the user data out of the last received NM message shall be copied to
Return code	
E_OK	No errors
E_NOT_OK	Retrieval of user data has failed
Functional Description	
Get user data from the last NM message received previously on the bus.	
Particularities and Limitations	
<ul style="list-style-type: none">> Service ID: see table 'Service IDs'> This function is non-reentrant.> This function is synchronous.> This function is called by NM Interface.> This function is available if User Data Enabled is ON (<code>FRNM_USER_DATA_ENABLED</code> is <code>STD_ON</code>).	
Call Context	
<ul style="list-style-type: none">> Task and interrupt level	

Table 4-10 FrNm_GetUserData

4.2.2.6.3 FrNm_GetPduData: Get NM PDU Data

Prototype	
Std_ReturnType FrNm_GetPduData (const NetworkHandleType NetworkHandle, const uint8 *nmPduData)	
Parameter	
NetworkHandle	Identification of the system channel
nmPduData	Pointer where the NM PDU Data shall be copied to
Return code	
E_OK	No errors
E_NOT_OK	Retrieval of NM PDU Data has failed
Functional Description	
Get the whole PDU data of the last received NM message.	
Particularities and Limitations	
<ul style="list-style-type: none"> > Service ID: see table 'Service IDs' > This function is non-reentrant. > This function is synchronous. > This function is called by NM Interface. > This function is available if Source Node Identifier Enabled is ON (FRNM_SOURCE_NODE_IDENTIFIER_ENABLED) is STD_ON, Node Detection Enabled is ON (FRNM_NODE_DETECTION_ENABLED is STD_ON) or User Data Enabled is ON (FRNM_USER_DATA_ENABLED is STD_ON). 	
Call Context	
<ul style="list-style-type: none"> > Task and interrupt level 	

Table 4-11 FrNm_GetPduData

4.2.2.7 Node Detection

4.2.2.7.1 FrNm_RepeatMessageRequest: Set Repeat Message Request Bit

Prototype	
Std_ReturnType FrNm_RepeatMessageRequest (const NetworkHandleType NetworkHandle)	
Parameter	
NetworkHandle	Identification of the system channel
Return code	
E_OK	No error
E_NOT_OK	Repeat Message Request has failed
Functional Description	
Set Repeat Message Request Bit for NM messages transmitted next on the bus. This service has no effect if the current mode is Bus Sleep Mode or Synchronize Mode or the configuration setting Repeat Message Bit Active is OFF.	
Particularities and Limitations	
<ul style="list-style-type: none"> > Service ID: see table 'Service IDs' > This function is non-reentrant. > This function is asynchronous. > This function is called by NM Interface. > This function is available if Passive Mode Enabled is OFF (<code>FRNM_PASSIVE_MODE_ENABLED</code> is <code>STD_OFF</code>) and if Node Detection Enabled is ON (<code>FRNM_NODE_DETECTION_ENABLED</code> is <code>STD_ON</code>). 	
Call Context	
<ul style="list-style-type: none"> > Task and interrupt level 	

Table 4-12 FrNm_RepeatMessageRequest

4.2.2.7.2 FrNm_GetNodeIdentifier: Get Source Node Identifier

Prototype	
<pre>Std_ReturnType FrNm_GetNodeIdentifier (</pre>	
const NetworkHandleType NetworkHandle,	
const uint8 *nmNodeIdPtr)	
Parameter	
NetworkHandle	Identification of the system channel
nmNodeIdPtr	Pointer where node identifier from the last received NM message is copied to
Return code	
E_OK	No error
E_NOT_OK	Retrieval of the node identifier from the last received NM message has failed
Functional Description	
Get the node identifier from the last received NM message.	
Particularities and Limitations	
<ul style="list-style-type: none">> Service ID: see table 'Service IDs'> This function is reentrant.> This function is synchronous.> This function is called by NM Interface.> This function is available if Source Node Identifier Enabled is ON (FRNM_SOURCE_NODE_IDENTIFIER_ENABLED is STD_ON).	
Call Context	
<ul style="list-style-type: none">> Task and interrupt level	

Table 4-13 FrNm_GetNodeIdentifier

4.2.2.7.3 FrNm_GetLocalNodeIdentifier: Get Local Source Node Identifier

Prototype

```
Std_ReturnType FrNm_GetLocalNodeIdentifier (  
    const NetworkHandleType NetworkHandle,  
    const uint8 *nmNodeIdPtr)
```

Parameter

NetworkHandle	Identification of the system channel
nmNodeIdPtr	Pointer where node identifier of the local node is copied to

Return code

E_OK	No error
E_NOT_OK	Getting of local node identifier has failed

Functional Description

Get the node identifier configured for the local node.

Particularities and Limitations

- > Service ID: see table 'Service IDs'
- > This function is reentrant.
- > This function is synchronous.
- > This function is called by NM Interface.
- > This function is available if Source Node Identifier Enabled is ON (FRNM_SOURCE_NODE_IDENTIFIER_ENABLED is STD_ON).

Call Context

- > Task and interrupt level

Table 4-14 FrNm_GetLocalNodeIdentifier

4.2.2.8 Remote Sleep Indication

4.2.2.8.1 FrNm_CheckRemoteSleepIndication: Check for Remote Sleep Indication

Prototype	
<pre>Std_ReturnType FrNm_CheckRemoteSleepIndication (const NetworkHandleType NetworkHandle, const boolean *nmRemoteSleepIndPtr)</pre>	
Parameter	
NetworkHandle	Identification of the system channel
nmRemoteSleepIndPtr	Pointer where check result of remote sleep indication shall be copied to
Return code	
E_OK	No error
E_NOT_OK	Checking of remote sleep indication has failed
Functional Description	
Check if remote sleep indication takes place or not.	
Particularities and Limitations	
<ul style="list-style-type: none">> Service ID: see table 'Service IDs'> This function is reentrant (but not for the same NetworkHandle).> This function is synchronous.> This function is called by NM Interface.> This function is available if Remote Sleep Indication Enabled is ON (FRNM_REMOTE_SLEEP_INDICATION_ENABLED is STD_ON).	
Call Context	
<ul style="list-style-type: none">> Task and interrupt level	

Table 4-15 FrNm_CheckRemoteSleepIndication

4.2.2.9 Bus Synchronization

4.2.2.9.1 FrNm_RequestBusSynchronization: Synchronization of Networks

Prototype

```
Std_ReturnType FrNm_RequestBusSynchronization (const NetworkHandleType  
NetworkHandle)
```

Parameter

NetworkHandle	Identification of the system channel
---------------	--------------------------------------

Return code

E_OK	No errors
------	-----------

Functional Description

Request bus synchronization. Since FlexRay is a synchronous bus, this service does not provide any functionality.

Particularities and Limitations

- > Service ID: see table 'Service IDs'
- > This function is reentrant.
- > This function is synchronous.
- > This function is called by NM Interface.
- > This function is available if Passive Mode Enabled is OFF (`FRNM_PASSIVE_MODE_ENABLED` is `STD_OFF`) and Bus Synchronization Enabled is ON (`FRNM_BUS_SYNCHRONIZATION_ENABLED` is `STD_ON`).

Call Context

- > Task level only

Table 4-16 FrNm_RequestBusSynchronization

4.2.2.10 NM Message Transmission Request

4.2.2.10.1 FrNm_Transmit: Spontaneous NM Message Transmission

Prototype	
Std_ReturnType FrNm_Transmit (PduIdType FrNmTxPduId, const PduInfoType *PduInfoPtr)	
Parameter	
FrNmTxPduId	Identification of the NM user data PDU
PduInfoPtr	Pointer to the Pdu data which shall be transmitted
Return code	
E_OK	No errors
Functional Description	
The FlexRay NM will not execute any functionality as no message outside the configured slots can be sent.	
Particularities and Limitations	
<ul style="list-style-type: none">> Service ID: see table 'Service IDs'> This function is reentrant.> This function is synchronous.> This function is called by PDU Router.> This function is available if Com User Data Support is ON (<code>FRNM_COM_USER_DATA_SUPPORT</code> is <code>STD_ON</code>).	
Call Context	
<ul style="list-style-type: none">> Task and interrupt level	

Table 4-17 FrNm_Transmit

4.2.2.11 Communication Control Service

4.2.2.11.1 FrNm_EnableCommunication

Prototype	
Std_ReturnType FrNm_EnableCommunication (const NetworkHandleType NetworkHandle)	
Parameter	
NetworkHandle	Identification of the system channel
Return code	
E_OK	No errors
E_NOT_OK	Enabling of NM PDU transmission ability has failed
Functional Description	
Enable the NM PDU transmission ability due to a ISO14229 Communication Control (28hex) service	
Particularities and Limitations	
<ul style="list-style-type: none">> Service ID: see table 'Service IDs'> This function is reentrant (but not for the same NM-channel).> This function is asynchronous.> This function is called by NM Interface.> This function is available if Com Control Enabled is ON (FRNM_COM_CONTROL_ENABLED is STD_ON).	
Call Context	
<ul style="list-style-type: none">> Task and interrupt level	

Table 4-18 FrNm_EnableCommunication

4.2.2.11.2 FrNm_DisableCommunication

Prototype

```
Std_ReturnType FrNm_DisableCommunication (const NetworkHandleType  
NetworkHandle)
```

Parameter

NetworkHandle	Identification of the system channel
---------------	--------------------------------------

Return code

E_OK	No errors
E_NOT_OK	Enabling of NM PDU transmission ability has failed

Functional Description

Disable the NM PDU transmission ability due to a ISO14229 Communication Control (28hex) service

Particularities and Limitations

- > Service ID: see table 'Service IDs'
- > This function is reentrant (but not for the same NM-channel).
- > This function is asynchronous.
- > This function is called by NM Interface.
- > This function is available if Com Control Enabled is ON (`FRNM_COM_CONTROL_ENABLED` is `STD_ON`).

Call Context

- > Task and interrupt level

Table 4-19 FrNm_DisableCommunication

4.2.2.12 Coordinator Synchronization Support

4.2.2.12.1 FrNm_SetSleepReadyBit

Prototype

```
Std_ReturnType FrNm_SetSleepReadyBit (  
    const NetworkHandleType NetworkHandle,  
    const boolean nmSleepReadyBit)
```

Parameter

NetworkHandle	Identification of the system channel
nmSleepReadyBit	Value written to Ready Sleep Bit in CBV

Return code

E_OK	No error
E_NOT_OK	Setting the Coordinator Sleep Ready bit has failed

Functional Description

Set the NM Coordinator Sleep Ready bit in the Control Bit Vector

Particularities and Limitations

- > Service ID: see table 'Service IDs'
- > This function is non-reentrant.
- > This function is synchronous.
- > This function is called by NM Interface.
- > This function is available if Coordinator Sync Support is ON (FRNM_COORDINATOR_SYNC_SUPPORT is STD_ON).

Call Context

- > Task level only

Table 4-20 FrNm_SetSleepReadyBit

4.2.2.13 Synchronized PNC Shutdown Support

4.2.2.13.1 FrNm_RequestSynchronizedPncShutdown

Prototype	
<pre>Std_ReturnType FrNm_RequestSynchronizedPncShutdown (const NetworkHandleType NetworkHandle, const PNCHandleType pncId)</pre>	
Parameter	
NetworkHandle	Identification of the system channel
pncId	PNC that needs to be shutdown
Return code	
E_OK	No error
E_NOT_OK	Request for a synchronized PNC shutdown has failed
Functional Description	
<p>Stores the PNC shutdown request and initiates a PNSR NM message that forwards the shutdown request.</p> <p>This function is only called on actively coordinated channels for a Top-Level PNC coordinator and intermediate PNC coordinator.</p>	
Particularities and Limitations	
<ul style="list-style-type: none"> > Service ID: see table 'Service IDs' > This function is reentrant for different channels. > This function is synchronous. > This function is called by NM Interface. 	
Call Context	
<ul style="list-style-type: none"> > Task and interrupt level 	

Table 4-21 FrNm_RequestSynchronizedPncShutdown

4.3 Services used by FRNM

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

Component	API
Det	Det_ReportError ¹⁰
Frlf	Frlf_GetNmVector ¹¹ Frlf_GetGlobalTime Frlf_GetState

¹⁰ Service only used if 'Dev Error Detect' is enabled

¹¹ Service only used if 'Hw Vote Enable' is enabled

Component	API
Nm	FrIf_Transmit ¹² Nm_BusSleepMode Nm_CarWakeUpIndication ¹³ Nm_CoordReadyToSleepCancellation ^{14,15} Nm_CoordReadyToSleepIndication ¹⁴ Nm_ForwardSynchronizedPncShutdown ¹⁶ Nm_NetworkMode Nm_NetworkStartIndication Nm_PduRxIndication ¹⁷ Nm_RemoteSleepCancellation ¹⁸ Nm_RemoteSleepIndication ¹⁸ Nm_RepeatMessageIndication ¹⁹ Nm_StateChangeNotification ²⁰ Nm_SynchronizationPoint ²¹ Nm_TxTimeoutException ¹²
PduR	PduR_FrNmTriggerTransmit ²² PduR_FrNmTxConfirmation ²² PduR_FrNmRxIndication ²³
SchM	SchM_Enter_FrNm_FRNM_EXCLUSIVE_AR EA_i SchM_Exit_FrNm_FRNM_EXCLUSIVE_ARE A_i for i = 0,...,6

Table 4-22 Services used by the FRNM

¹² Service only used if ‘Passive Mode’ is disabled¹³ Service only used if ‘Car Wake Up Rx Enabled’ is enabled¹⁴ Service only used if ‘Coordinator Sync Support’ is enabled¹⁵ Service is a Vector extension to the Nm module¹⁶ Service only used if ‘Synchronized PNC Shutdown Enabled’ is enabled.¹⁷ Service only used if ‘Pdu Rx Indication Enabled’ is enabled¹⁸ Service only used if ‘Remote Sleep Indication Enabled’ is enabled¹⁹ Service only used if ‘Node Detection Enabled’ is enabled²⁰ Service only used if ‘State Change Notification Enabled’ is enabled²¹ Service only used if ‘Coordinator Support Enabled’ is enabled and ‘Synchronization Point Enabled’ is enabled²² Service only used if ‘Com User Data Support’ is enabled.²³ Service only used if at least one of the features ‘Pn Eira Calc Enabled’ or ‘Pn Era Calc Enabled’ are enabled.

4.4 Callback Functions

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

4.4.1 Callback Functions from FlexRay Interface

4.4.1.1 FrNm_TxConfirmation: NM PDU Transmission Confirmation

Prototype	
<code>void FrNm_TxConfirmation (PduIdType FrNmTxPduId)</code>	
Parameter	
FrNmTxPduId	Identification of the transmitted PDU. The NM channel on which the PDU was transmitted has to be encoded in this parameter.
Return code	
-	-
Functional Description	
Notification that a NM PDU has been successfully transmitted.	
Particularities and Limitations	
<ul style="list-style-type: none"> > Service ID: see table 'Service IDs' > This function is reentrant (but not for the same <code>FrNmTxPduId</code>). > This function is asynchronous. > This service may only be called by FlexRay Interface. > Not available for passive nodes (<code>FRNM_PASSIVE_NODE_ENABLED</code> is <code>STD_ON</code>). 	
Call context	
<ul style="list-style-type: none"> > Task and interrupt level. 	

Table 4-23 FrNm_TxConfirmation

4.4.1.2 FrNm_RxIndication: NM PDU Reception Indication

Prototype	
<code>void FrNm_RxIndication (PduIdType FrNmRxPduId, const PduInfoPtr* PduInfoPtr)</code>	
Parameter	
FrNmRxPduId	Identification of the received PDU. The NM channel on which the PDU was received has to be encoded in this parameter.
PduInfoPtr	Pointer to a <code>PduInfoType</code> constant where a pointer to the FlexRay NM SDU data and length is stored.
Return code	
-	-
Functional Description	
Notification that a NM PDU has been received.	

Particularities and Limitations

- > Service ID: see table 'Service IDs'
- > This function is reentrant (but not for the same `FrNmRxPduId`).
- > This function is asynchronous.
- > This service may only be called by FlexRay Interface.

Call context

- > Task and interrupt level

Table 4-24 FrNm_RxIndication

4.4.1.3 FrNm_TriggerTransmit: NM PDU pre-transmit function

Prototype

```
void FrNm_TriggerTransmit (PduIdType FrNmTxPduId,
                           PduInfoType* PduInfoPtr )
```

Parameter

<code>FrNmTxPduId</code>	Identification of the PDU that is transmitted next. The NM channel on which the PDU is transmitted has to be encoded in this parameter.
<code>PduInfoPtr</code>	Pointer to a <code>PduInfoType</code> variable where the NM PDU data and length has to be copied to.

Return code

-	-
---	---

Functional Description

Notification that a NM message will be transmitted next and the corresponding PDU data has to be copied. This service is only called for NM messages that are sent in decoupled mode.

Particularities and Limitations

- > Service ID: see table 'Service IDs'
- > This function is reentrant (but not for the same `FrNmTxPduId`).
- > This function is synchronous.
- > This service may only be called by FlexRay Interface.
- > Not available for passive nodes (`FRNM_PASSIVE_NODE_ENABLED` is `STD_ON`).

Call context

- > Task and interrupt level

Table 4-25 FrNm_TriggerTransmit

4.4.2 Callback Function from FlexRay State Manager

4.4.2.1 FrNm_StartupError: FlexRay Synchronization Loss Indication

Prototype

```
void FrNm_StartupError (const NetworkHandleType NetworkHandle)
```

Parameter

NetworkHandle	Identification of the system channel
---------------	--------------------------------------

Return code

-	-
---	---

Functional Description

Notify FlexRay NM that a long-term synchronization loss has been detected.

Particularities and Limitations

- > Service ID: see table 'Service IDs'
- > This function is non-reentrant.
- > This function is synchronous.
- > This service may only be called by FlexRay State Manager.

Call Context

- > Task level only

Table 4-26 FrNm_StartupError

5 Glossary and Abbreviations

5.1 Glossary

Term	Description
Confirmation	Notification by the data link layer on asynchronous successful transmission of a message.
FlexRay Channel	One FlexRay bus contains two FlexRay Channels (A and B) which cannot be handled independently for the NM. Therefore a NM Cluster covers both channels.
Identifier	Identifies a message.
Indication	Notification by the data link layer on asynchronous reception of a message.
Message	One or more signals are assigned to each message.
NM Channel	Logical communication path associated with one NM cluster.
NM Cluster	Instance of the NM to handle one physical (FlexRay) bus.
NM Data	Consists of NM CBV, NM NID and NM User Data.
NM Vote	NM voting information. This information is realized in the static segment by a separate bit in the NM PDU. In the dynamic segment the presence / non-presence of the NM PDU corresponds to the vote.
Signal	Signals describe the significance of the individual data segments within a message. Typically bits, bytes or words are used for data segments but individual bit combinations are also possible. In the database, each data segment is assigned a symbolic name, a value range, a conversion formula and a physical unit, as well as a list of receiving nodes.

Table 5-1 Glossary

5.2 Abbreviations

Abbreviation	Description
API	Application Programming Interface
AUTOSAR	Automotive Open System Architecture
CAN	Controller Area Network
CBV	Control Bit Vector
Com	Communication (AUTOSAR Basic Software Module)
ComM	COM Manager
CRI	Partial Network Cluster Request Information
DET	Development Error Tracer
DEM	Diagnostic Event Manager
DLC	Data Length Code (Number of data bytes of a CAN message)
ECU	Electronic Control Unit

EIRA	External Internal Requests Aggregated
ERA	External Requests Aggregated
FIBEX	Field Bus Exchange
FrIf	FlexRay Interface
ID	Identifier (of a CAN message)
I-PDU	Interaction Layer Protocol Data Unit
LIN	Local Interconnect Network
NID	Node IDentifier
NM	Network Management
PDU	Protocol Data Unit
PDUR	PDU Router
PN	Partial Network / Partial Networking
PNSR	Partial Network Shutdown Request
RAM	Random Access Memory
ROM	Read Only Memory
SRS	System Requirements Specification (used for AUTOSAR documents)
SWS	Software Specification (used for AUTOSAR documents)
UDP	User Datagram Protocol

Table 5-2 Abbreviations

6 Contact

Visit our website for more information on

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

www.vector.com