VECTOR

MICROSAR Classic Communication Manager

Technical Reference

Version 13.02.00

Authors	vismih, vislsi, vispps, smarcelin
Status	Released

Document Information

History

Author	Date	Version	Remarks
vismih	2012-08-07	1.00.00	Initial creation
vismih	2013-01-31	1.01.00	> Updated AUTOSAR Architecture, chapter 1.1 > Updated description of ComM_CurrentChannelRequest, chapter 5.1.3.1
vismih	2013-05-15	1.02.00	> Added configuration variant Post-Build Loadable, chapter 2.1.2.2 (ESCAN00064954) > Information from chapter ‘AUTOSAR Standard Compliance’ is moved to chapter ‘Features’ > Updated chapter ‘Critical Sections’ > Removed chapter ‘Compiler Abstraction and Memory Mapping’ > Updated chapter ‘Partial Network States’
vismih	2013-09-27	2.00.00	> Updated chapter ‘Partial Network States’ (ESCAN00069988) > Updated chapters ‘Mode Limitation’, ‘ComM_PreventWakeUp’, ‘ComM_LimitChannelToNoComMode’, ‘ComM_LimitECUToNoComMode’ (ESCAN00068896 , ESCAN00068902) > Support EthSM as lower layer of COMM, see chapters 2.4 and 4.3 (ESCAN00069043) > Updated description of Sender Receiver Interface ‘ComM_CurrentMode’ (ESCAN00070321). > Added notes for usage of LIN NM and J1939 NM in chapters 2.6.1 and 2.6.3 (ESCAN00071341). > Updated Include Structure to reflect Post Build Loadable (ESCAN00064954). > Added a Caution box to chapter 2.5.1 regarding Partial Network TX EIRA signals (ESCAN00071527).
vismih	2014-02-07	2.01.00	> ESCAN00072763 Added DET error COMM_E_DIAGNOSTIC_NOT_SUPPORTED in chapters 2.7.1, 4.4.6 and 4.4.7 > ESCAN00074243 Table 2-5 sorted by ID, description of the Port Interface ComM_CurrentMode moved to chapter 5.1.2 ‘Mode Switch Interface’.

vismih	2014-06-04	3.00.00	> Improved description of ComM_CurrentChannelRequest (ESCAN00075361). > Improved description of ComMPncPrepareSleepTimer in chapter 2.5.1 (ESCAN00075422). > Added chapter 2.1.3.2 (ESCAN00076076). > Adapted chapter 5.1.1.1.3 (ESCAN00074321).
vismih	2014-10-02	3.01.00	> Added support of Bus Type INTERNAL, refer to chapter 2.5.2 (ESCAN00076859) > Added Timer-based shutdown synchronization via Silent state, refer to chapter 2.1.2.3 and Figure 2-1 COMM channel state machine (ESCAN00076774) > Updated include structure in 3.2 (ESCAN00078688) > Extended Partial Network Cluster ID support, refer to chapter 2.1.2 (ESCAN00076852) > NvM support made optional, refer to chapter 2.1.2 (ESCAN000772069) > Added internal function service IDs (ESCAN00076325)
vismih	2015-03-25	4.00.00	> Support of channel-specific Minimum Full Com Mode timer, chapters 2.1.2.4 and 2.4 (ESCAN00082062)
vismih	2015-08-12	5.00.00	> ‘Post-Build Loadable’ is used as standard MICROSAR feature name, chapter 2.1.2.2 > Service Port names do not contain ComM user and channel identifiers anymore, see chapter 5 (ESCAN00084462) > Added Pnc to Channel Routing Limitation, see chapter 2.1.2.5 (ESCAN00083603)
vismih	2016-01-12	6.00.00	> Improved description of 2.1.1 Deviations (ESCAN00087416)
vismih	2016-02-26	7.00.00	> Added support of Extended RAM Check, see chapter 2.1.2.6 (ESCAN00089104) > Improved description of 3.3 Critical Sections (ESCAN00088182)
vismih	2016-05-24	7.00.01	> Added API description of ComM_Nm_StateChangeNotification and ComM_ComCbk_<SignalName> (ESCAN00090164)

vismih	2016-08-15	7.01.00	> Added APIs ComM_GetDcmRequestStatus and ComM_GetMinFullComModeTimerStatus (ESCAN00091481) > Provided more details on Nm Variant PASSIVE in chapters 2.1.2 and 2.6.1
vismih	2016-11-09	8.00.00	> Added Reset after Forcing NO_COM functionality in chapter 2.6.2.1
vislsi	2019-07-26	9.00.00	> Changed the type of ComM_UserHandleType in chapter 4.1
vislsi	2019-08-21	10.00.00	> Added Nm Variant LINSLAVE in chapters 2.1, 2.4 and 4.4
vislsi	2019-11-15	10.01.00	> Added support for managed and managing channels in chapters 2.4, 2.5.1 and 2.6.1
vispps	2019-12-06	10.02.00	> Added Support for ComM0PncVectorAvoidance, see chapters 2.1 and 2.5.2 > NMM-994: Added further information about Partial Network Gateway and Routing, see chapter 2.5.2
vislsi/vispps	2020-01-29	11.00.00	> Added support for Gateway Type NONE for coordinated PNC(s), see chapters 2.1.2 and 2.5.1 > Added chapter for Start of Transmission and Reception (ESCAN00104287), see chapter 2.1.3.3 > Added Support for API ComM_EcuM_PNCWakeUpIndication (ESCAN00092764), see 2.5.1 and 4.4.3 > Added further information about PNC activation request received on channel with gateway type NONE, see chapter 2.5.2
vislsi	2020-10-14	11.00.01	> Extended description in chapter Mode Limitation to NO_COM, see chapter 2.6.2.12.6.2 > Extended further description for API ComM_LimitChannelToNoComMode, see chapter 4.2.14
vislsi	2021-04-05	12.00.00	> Updated Chapters 2.3, 2.7.1, 3.1.2 and 4.2.3 > Added Chapters 4.2.2, 4.2.4, 4.2.25 and 2.2

smarcelin/vispps	2021-07-20	12.01.00	> Added support for ComM_GetCurrentPNCComMode, see chapters 2.7.1, 2.7.1.1 and 4.2.12 > Updated chapters 4.2.10, 4.2.11 and 4.2.22 (ESCAN00109709) > Updated chapter 3.3 (ESCAN00109218) > Added new critical sections for Multi-Partition, see chapter 3.3 > Added integration information for Multi-Partition, see chapter 3.5
smarcelin	2021-09-15	12.01.01	> Updated chapter 2.5.1 (ESCAN00105892)
vispps	2021-10-21	13.00.00	> Added support Top Down Service Configuration for ComMgrUserNeeds. Updated chapter 2.1.
smarcelin	2021-11-24	13.01.00	> Updated chapter 4.2.12.
smarcelin/vispps	2022-01-18	13.02.00	> Added support for synchronized PNC shutdown. Added chapters 2.5.3 and 4.4.14. Updated chapters 2.1 and 2.7.1. > Updated chapter 2.1.1 (ESCAN00107203).

Reference Documents

No.	Source	Title	Version
[1]	AUTOSAR	Specification of Communcation Manager	R4.0.3
[2]	AUTOSAR	Specification of Development Error Tracer	R4.0.3
[3]	AUTOSAR	Specification of BSW Module List	R4.0.3
[4]	AUTOSAR	Specification of Ethernet State Manager	R4.1.1
[5]	AUTOSAR	Specification of UDP Network Management	R4.1.1
[6]	Vector	TechnicalReference Lin Network Management	see delivery

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

Contents

Illustrations

Tables

1 Introduction

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

Supported AUTOSAR Release:	4	
Supported Configuration Variants:	pre-compile, post-build-loadable	
Vendor ID:	COMM_VENDOR_ID	30 decimal (= Vector-Informatik, according to HIS)
Module ID:	COMM_MODULE_ID	12 decimal (according to ref. [3])

The Communication Manager is a resource manager, which encapsulates the control of the underlying communication services.

The purpose of the COMM module is:

> Coordinating different wake-up events independent of the used bus system.

> Providing the concept of user to request Communication Modes. Coordinating requests of multiple independent users.

> Controlling of more than one communication bus channel of an ECU by implementing a channel state machine for every channel.

> Simplifying the resource management by allocating all resources which are necessary to start or shutdown communication.

> Simplifying the handling of the underlying communication stack (e.g. network management handling).

> Providing mode inhibition functionality to limit the communication capabilities of the ECU.

1.1 Architecture Overview

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

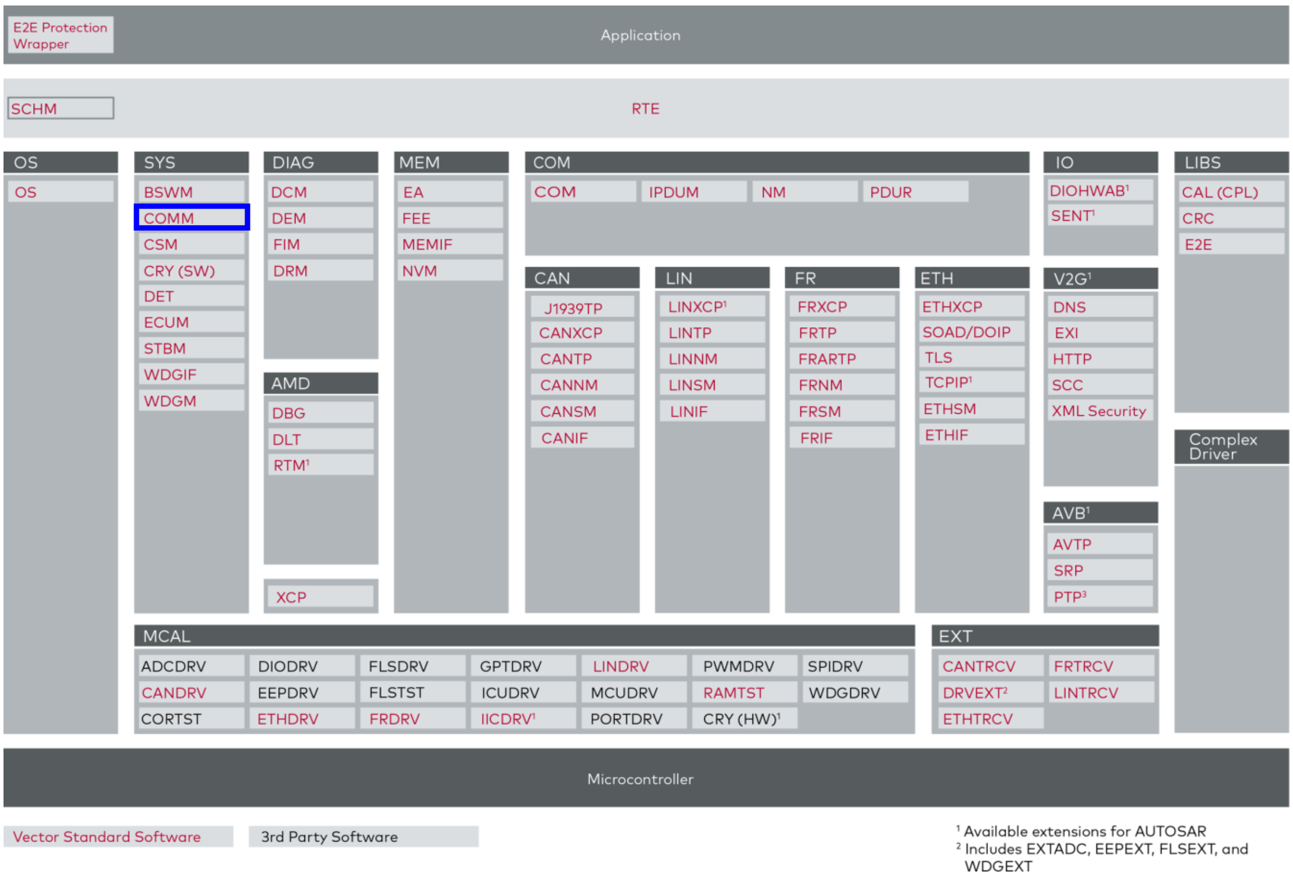

Figure 1-1 AUTOSAR 4.x Architecture Overview

The next figure shows the interfaces to adjacent modules of the COMM. These interfaces are described in chapter 3.

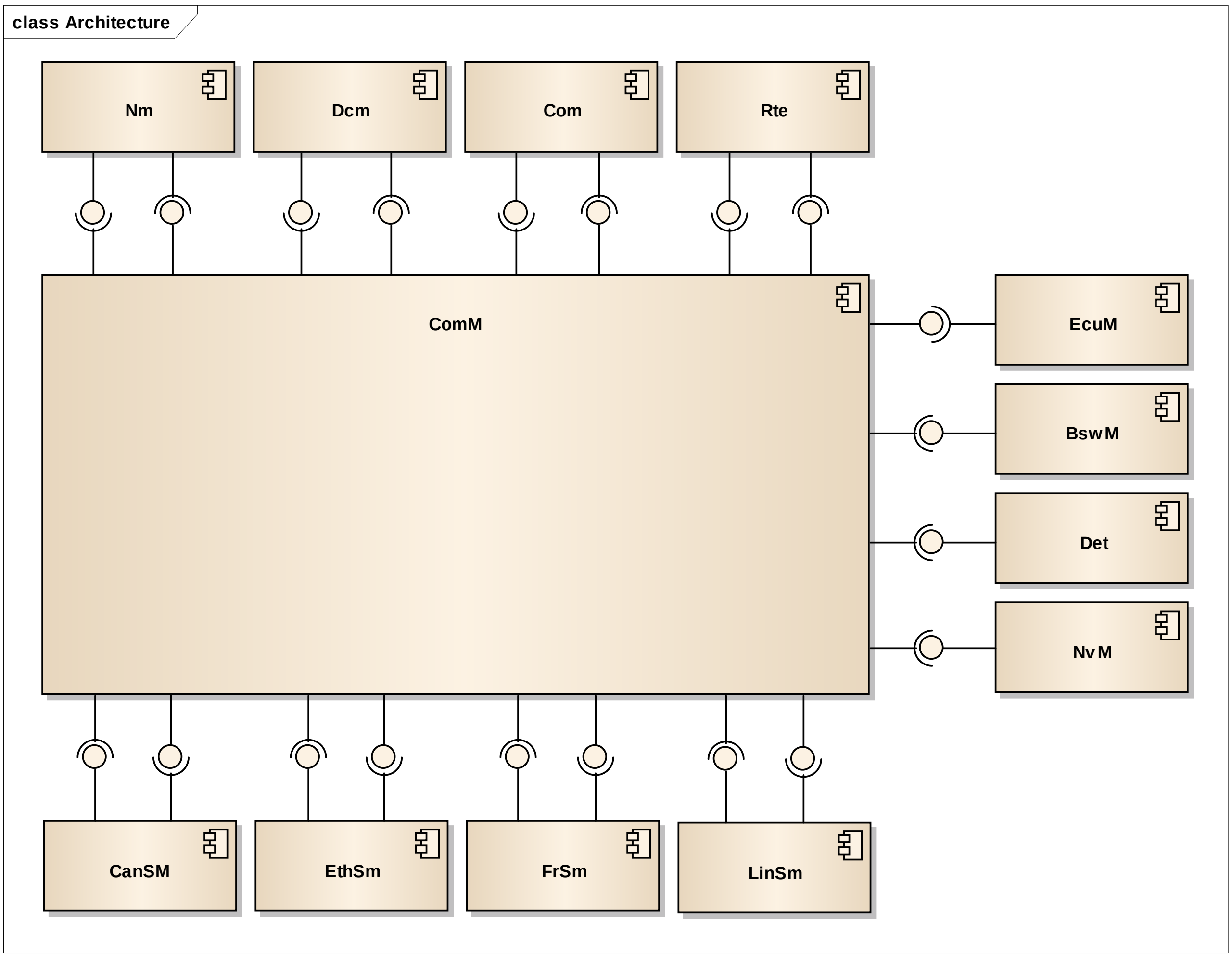

Figure 1-2 Interfaces to adjacent modules of the COMM

Applications do not access the services of the BSW modules directly. They use the service ports provided by the BSW modules via the RTE. The service ports provided by the COMM are listed in chapter 5 and are defined in [1].

2 Functional Description

2.1 Features

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

The AUTOSAR standard functionality is specified in [1], 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

Refer to the chapter 2.1.1 for further information on not supported features.

Vector Informatik provides further COMM 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 [1] are supported:

Supported AUTOSAR Standard Conform Features
Communication Control Handling
Synchronous wake up of channels
NM Variant Handling: FULL
NM Variant Handling: LIGHT
NM Variant Handling: NONE
NM Variant Handling: PASSIVE
NM Variant Handling: LINSLAVE
Mode Limitation No Communication
Bus Wake-up Inhibition
Service Port: ComM_UserRequest
Service Port: ComM_ECUModeLimitation
Service Port: ComM_ChannelWakeUp
Service Port: ComM_ChannelLimitation
Service Port: ComM_CurrentMode
Service Port: ComM_CurrentChannelRequest
Storage of non-volatile values
Partial Network Cluster Management

Supported AUTOSAR Standard Conform Features
Detection and notification of development errors to DET
COMM bus type INTERNAL
Reset after Forcing NO_COM functionality, see chapter 2.6.2.1
Callback Function: ComM_BusSM_BusSleepMode (AUTOSAR 4.4.0)
Support for managed and managing channels
Support for ComM0PncVectorAvoidance (AUTOSAR 4.4.0)
Support for Top Down Service Configuration for ComMgrUserNeeds
Support for Synchronized PNC Shutdown, see chapter 2.5.3

Table 2-1 Supported AUTOSAR standard conform features

2.1.1 Deviations

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

Category	Description	ASR Version
Config	ComMDirectUserMapping	4.0.3
Config	ComMUserEcucPartitionRef	4.0.3
Config	ComMUserPerPnc (COMM does not support PNCs without user assigned to it)	4.0.3
Functional	Version checking (COMM does not perform Inter Module version checks)	4.0.3 < 4.1.2
Config	ComMUser-PNC mapping (COMM does not restrict the mapping of a ComMUser to a PNC and in addition to a ComMChannel which is already referenced by the PNC. Please note that this mapping does not affect the User-Requests for the ComMChannel, since the ComMChannel would always be requested and released at the same time for both the ComMUserPerPnc and the ComMUserPerChannel.)	4.0.3

Table 2-2 Not supported AUTOSAR standard conform features

2.1.1.1 Variant Post-Build

Instead of the Configuration Variant Post-Build, the Variant Post-Build-Loadable is supported.

2.1.2 Additions/ Extensions

The following features are provided beyond the AUTOSAR standard:

Features Provided Beyond The AUTOSAR Standard
Enabling or disabling of User Mode Notification per COMM user (ComMUserModeNotification)
Optional inclusion of a user configuration file (ComMUserConfigurationFile)
Development Error Reporting is extended by COMM_E_NOSUPPORTED_MODECHANGE, COMM_E_ERROR_IN_PROV_SERVICE, COMM_E_DIAGNOSTIC_NOT_SUPPORTED
Memory Initialization

Features Provided Beyond The AUTOSAR Standard
Post-Build Loadable, see chapter 2.1.2.2
As a compatibility feature, the Client-Server-Interface ComM_ChannelWakeUp can omit the operation GetInhibitionStatus, see chapter 5.1.1.1.3
Possibility to assign COMM users to COMM channels with Nm variant PASSIVE. Communication Requests of such users will be ignored. A possible use case is triggering a runnable via RTE mode switch interface ComM_CurrentMode.
Timer-based shutdown synchronization via Silent state, see chapter 2.1.2.3.
Partial Network Cluster ID counting is supported accordingly to Autosar version 4.0.x and 4.1.x as well. Refer to the description of parameter ‘Pnc Id Counting’ for more information.
NvM support is optional when using Mode Limitation. Refer to the description of parameter ‘Global NvM Block Descriptor’ for more information.
MICROSAR Identity Manager using Post-Build Selectable
Support of channel-specific Minimum Full Com Mode timer, see chapter 2.1.2.4
Pnc to Channel Routing Limitation
Extended RAM Check, see chapter 2.1.2.6
Gateway type NONE for coordinated PNC(s)

Table 2-3 Features provided beyond the AUTOSAR standard

2.1.2.1 Memory Initialization

Not every start-up code of embedded targets and neither CANoe provide initialized RAM. It thus may happen that the state of a variable that needs initialized RAM may not be set to the expected initial value. Therefore an explicit initialization of such variables has to be provided at start-up by calling the additional function `ComM_InitMemory`.

For more information refer to chapter 2.3 ‘Initialization’.

2.1.2.2 Post-Build Loadable

In the Variant Post-Build-Loadable, the configuration parameters ‘ComMChannelPerPnc’ and ‘ComMUserPerPnc’ are also changeable during the post-build phase as addition to the post-build-changeable parameter ComMPncEnabled required by [1].

The following use cases are supported in post-build phase in addition to [1]:

> Assign a non-coordinated PNC to another channel on an ECU with multiple channels.

> Assign a coordinated PNC to other channels.

> Remove or add one or more users to a PNC. It is allowed that a user is not assigned to any PNC anymore.

There are following limitations to be taken into account:

> Coordination type of PNCs cannot be changed in post-build phase. If a PNC was coordinated in pre-compile phase it shall remain coordinated in post-build phase and vice versa.

> If changing the assignment of PNC to channels, the PNC signal configuration made in pre-compile phase (parameter ‘ComMPncComSignal’) must reference the channels, which are added in post-build phase.

- PNCs cannot be added or removed in post-build phase.
- Each PNC shall be assigned to at least one channel and to at least one user.
- If a COMM user was assigned to one or more channels in pre-compile phase, it cannot be assigned to PNCs in post-build phase and vice versa.
- COMM users cannot be created or deleted in post-build phase.

2.1.2.3 Timer-based Shutdown Synchronization via Silent State

'Nm Light Silent Timeout' timer specifies the time duration spent in the state COMM_SILENT_COMMUNICATION after leaving COMM_FULL_COM_READY_SLEEP state and before entering COMM_NO_COMMUNICATION state. This is similar to the Prepare Bus Sleep Phase when Network Management is used. This timer is only available for channels with Bus Type CAN and Nm Variant LIGHT.

2.1.2.4 Channel-specific Minimum Full Com Mode Timer

The optional channel-specific parameter 'TMin Full Com Mode Duration Of Channel' is used to initialize the Minimum Full Com Mode timer of a channel. It specifies the minimum time duration, spent in the COMM_FULL_COMMUNICATION sub-state COMM_FULL_COM_NETWORK_REQUESTED. The parameter is only available for channels with Nm Variants LIGHT and FULL.

If the channel has Nm Variant LIGHT:

- This parameter is used instead of the global 'TMin Full Com Mode Duration'.
- The Minimum Full Com Mode timer is started when entering the state COMM_FULL_COMMUNICATION.
- The Minimum Full Com Mode timer is cancelled if a user or Dcm requests communication.

If the channel has Nm Variant FULL:

- It is recommended to use this parameter if the corresponding BusNm does not support the Repeat Message Time functionality (e.g. NmOsek).
- It is not recommended to use this parameter if the corresponding BusNm supports the Repeat Message Time functionality (e.g. CanNm, FrNm or UdpNm).
- The Minimum Full Com Mode timer is started when entering the state COMM_FULL_COMMUNICATION.
- The Minimum Full Com Mode timer cannot be cancelled.

2.1.2.5 Pnc to Channel Routing Limitation

This feature allows a selective limitation of Partial Network Routing at runtime.

The feature is de-activated per default. In this case ComM will route Partial Network requests to all channels mapped to the Partial Network according to AUTOSAR specification [1].

If the feature is activated, it is possible to limit the routing of Partial Network requests on particular channels using the API ComM_LimitPncToChannelRouting(). The Routing Limitation can be applied to channels with both active and passive gateway (coordination) types. There are three states of Routing Limitation on a channel that are described in the Table 2-4.

State of Partial Network Routing Limitation on the channel	GW routes PNC requests to the channel	GW keeps the channel awake
Disabled in one of the following cases: > Disabled temporarily as long as a ComM user mapped to the channel (not to PNC) requests FULL_COM or > Disabled temporarily as long as ERA signal containing a PNC request is received on the channel or > A PNC is requested by a ComM user or by another ECU and the routing of this PNC to the channel is **not** limited.	■	■
Partly disabled if > None of above applies and > As long as Network Management is in state 'Repeat Message' on the channel (e.g. after receiving Nm message in state 'Prepare Bus Sleep').	■	
Enabled if none of the above applies.		

Table 2-4 States of Routing Limitation on a channel

The states of Routing Limitation determine whether the GW routes PNC requests to a particular channel and whether the GW keeps the channel awake by sending its own Nm message. The basic rule is that the PNC requests are to be routed to a channel if the GW sends its Nm message on it.

The Routing Limitation states are exclusive and have the following meaning:

> If Routing Limitation is **disabled** on a channel, ComM will keep the channel awake and route the request to it. Nm will set the corresponding bits to 1 within PNC vector in the Nm message sent by the GW.

> Otherwise if Routing Limitation is **partly disabled** on a channel, ComM will not keep the channel awake but will route the request to it. Nm will set the corresponding bits to 1 within PNC vector in the Nm message sent by the GW.

> Otherwise if Routing Limitation is **enabled** on a channel, ComM will not keep the channel awake and there is no Nm messages sent by the GW.

The feature introduces an additional condition for a PNC to enter the state PNC_REQUESTED. If a ComM user mapped to the PNC requests FULL_COM, the PNC is allowed to enter PNC_REQUESTED if at least one channel mapped to the PNC has the Routing Limitation **disabled** or **partly disabled**. If all channels have the Routing Limitation **enabled**, the request is stored but inhibited.

The following use cases are aimed to illustrate the rules described above:

- PNC user requests FULL_COM, but Routing Limitation is **enabled** on all channels mapped to the PNC. The request is stored but not granted.
- PNC user requests FULL_COM and there is at least one channel mapped to the PNC having Routing Limitation **disabled** or **partly disabled**. ComM will execute the request and PNC will enter PNC_REQUESTED state. ComM will notify BswM.
- Requests via ERA=1 are always granted because Routing Limitation is **disabled** temporarily on the channel where ERA was received as long as the value of ERA is 1.
- Routing Limitation on a channel can be **disabled** or **partly disabled** while the channel is mapped to a PNC which is in state PNC_REQUESTED. Nm will set corresponding bits to 1 within PNC vector of the Nm message sent by the GW.
- Routing Limitation on a channel can be **enabled** while the channel is mapped to a PNC which is in state PNC_REQUESTED. GW will stop sending the Nm message on the channel. Other ECU's on the channel will release the PNC due to timeout of 'Pn Reset Time' in Nm.
- If a PNC is in state PNC_REQUESTED because a user requests FULL_COM and Routing Limitation is **enabled** on all channels, the PNC will enter PNC_READY_SLEEP state.

Caution
It is ensured that the content of PNC vector is consistent among all Nm messages that GW sends on particular channels. The content of PNC vector considers the mapping of Partial Networks to channels defined in the configuration. Therefore the content of PNC vector can differ on channels if there are PNCs that are not mapped to all channels.

Caution
The State Change Indication callback must be configured within Nm Interface and the related BusNm modules:
- 'State Change Ind Enabled' functionality must be activated in each related BusNm.
- 'Callbacks Prototype Header' of Nm Interface must be set to 'ComM_Nm.h'.
- 'State Change Indication Callback' of Nm Interface must be set to ComM_Nm_StateChangeNotification.

2.1.2.6 Extended RAM Check

ComM supports Extended RAM check of the CAN registers and Message Boxes. If the feature is activated, ComM evaluates RAM Check status before starting communication on a CAN channel. If Extended RAM Check fails on a CAN channel communication is not started.

2.1.3 Limitations

2.1.3.1 Non-volatile Data Handling

COMM uses only the NVM global block descriptor to handle the COMM non-volatile data.

2.1.3.2 Assignment of Users to Channels and PNCs

COMM does not support assigning a COMM user to Channel(s) and PNC(s) at the same time. Instead, it is recommended to create two COMM users in this case, assigning the first one to Channel(s) and the second one to PNC(s).

2.1.3.3 Start of Transmission and Reception

When COMM_FULL_COMMUNICATION is requested by ComM or by EcuM (Wakeup), the BusSM is requested to switch to full communication.
When the BusSM indicates the successful transition to full communication, then the ComM requests BusNM either Actively or Passively.

The above scenario doesn't hold true sometimes because usually the BusNM is faster than the BusSM and the NM will try to send NM messages on the bus, which will further start the NM timers before the BusSM reaches full communication. As NM messages are resposnisble for waking up the other nodes, this behavior could result in loss of some messages and not waking up the other nodes as the BusSM is still not in full communication.

2.2 Multi-Partition Handling

In case of Multi-Partition, the ComM module consists of multiple instances which are assigned to different partitions. These instances are divided into one master partition and one or multiple satellite partitions.
The master partition is defined as the partition where the ComM module is assigned to. This
partition provides the main functions for each channel and handles the data synchronization with the satellite partitions.
The satellite partitions are defined as the partitions where at least one ComM channel is assigned to. Each channel provides its own satellite main function where the
forwarding mechanism of services towards the Nm is performed.

Note

Even though the basic software (and the Com-Stack in particular) is distributed across several partitions, ComM and Nm Masters should reside in the same partition in order to keep the mode interfaces between the two modules simple.

2.2.1 Limitations

2.2.1.1 Request handling

All the calls to the ComM from the application or BswM must be received on the master partition.

2.2.1.2 Pnc Wake-up Indication

PNC wakeup indication is currently not supported with Multi-Partition.

2.3 Initialization

Before calling any other functionality of the COMM module, the module must be initialized. The initialization is realized by calling the intialization functions ComM_PreInit(), ComM_Init() and ComM_PostInit() successively.

For API details refer to chapter 4.2.3 ‘ComM_Init’, chapter 4.2.2 ‘ComM_PreInit’ and chapter 4.2.4 ‘ComM_PostInit’.

Note

The pre-initialization function ComM_PreInit must only be called once. It must be called before ComM_Init is called on any partition. In case of Multi-Partition, each instance of the ComM module must be initialized by calling ComM_Init on each partition where a ComM instance is located. The post-initialization function ComM_PostInit must only be called after all ComM instances are initialized, i.e. ComM_Init was called on every ComM partition. The ComM_PostInit must be called on the ComM master partition.

The COMM module assumes that some variables are initialized with certain values at start-up. As not all embedded targets support the initialization of RAM within the start-up code the COMM module provides the function `ComM_InitMemory()`. This function has to be called during start-up and before `ComM_PreInit()` is called. Refer also to chapter 2.1.2.1.

For API details refer to chapter 4.2.1 ‘ComM_InitMemory’’.

2.4 States

Figure 2-1 shows the COMM state machine, which consists of three main states representing abstracted status of communication capabilities per channel. These states correspond to the Communication Modes, which are in focus of the users’ interests:

> COMM_NO_COMMUNICATION,

> COMM_SILENT_COMMUNICATION,

> COMM_FULL_COMMUNICATION.

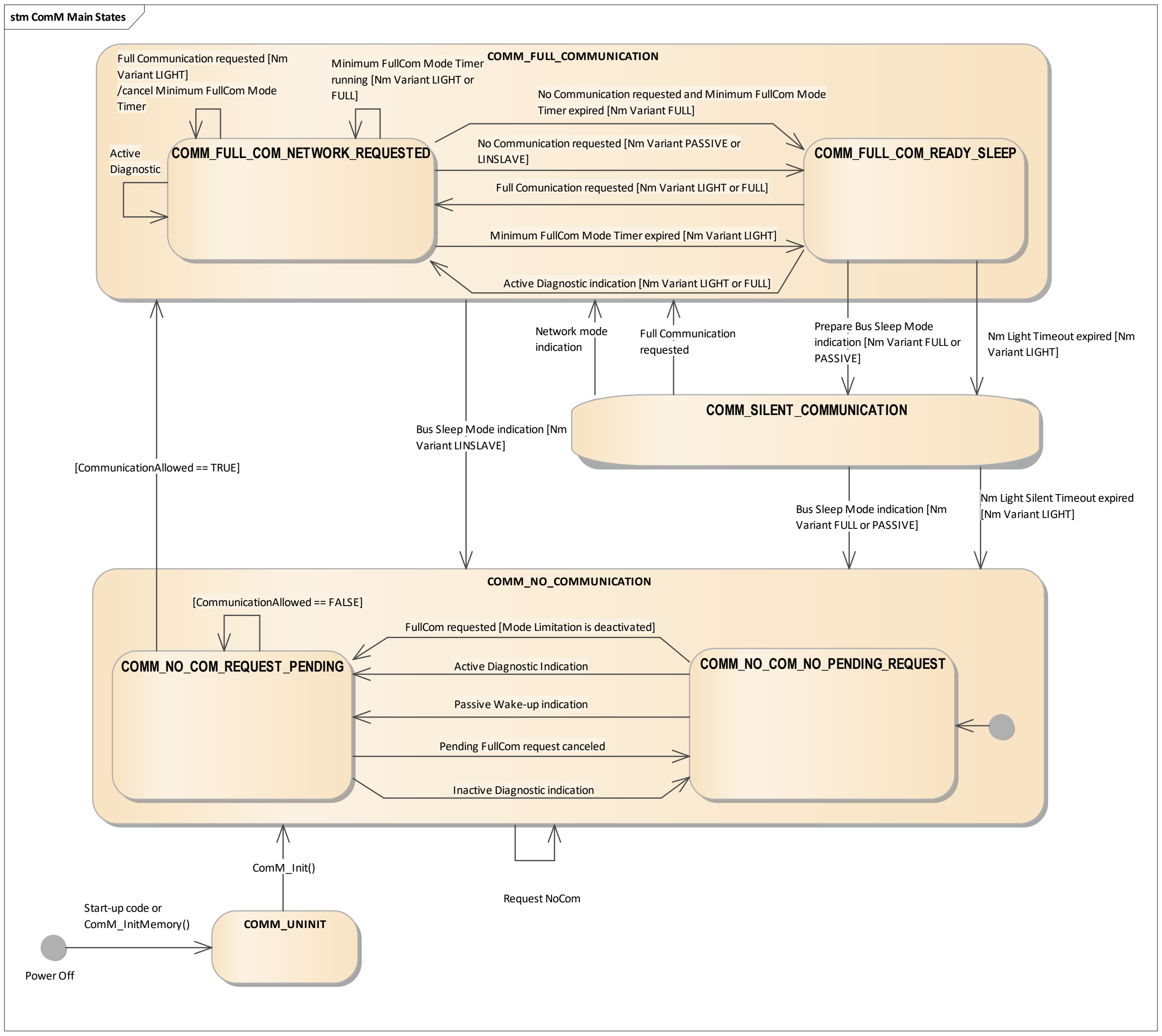
stm ComM Main States
COMM_FULL_COMMUNICATION
Full Communication requested [Nm Variant LIGHT] /cancel Minimum FullCom Mode Timer
Minimum FullCom Mode Timer running [Nm Variant LIGHT or FULL]
No Communication requested and Minimum FullCom Mode Timer expired [Nm Variant FULL]
Active Diagnostic
COMM_FULL_COM_NETWORK_REQUESTED
No Communication requested [Nm Variant PASSIVE or LINSLAVE]
COMM_FULL_COM_READY_SLEEP
Full Comunication requested [Nm Variant LIGHT or FULL]
Minimum FullCom Mode Timer expired [Nm Variant LIGHT]
Active Diagnostic indication [Nm Variant LIGHT or FULL]
Network mode indication
Full Communication requested
Prepare Bus Sleep Mode indication [Nm Variant FULL or PASSIVE]
Nm Light Timeout expired [Nm Variant LIGHT]
COMM_SILENT_COMMUNICATION
Bus Sleep Mode indication [Nm Variant LINSLAVE]
[CommunicationAllowed == TRUE]
Bus Sleep Mode indication [Nm Variant FULL or PASSIVE]
Nm Light Silent Timeout expired [Nm Variant LIGHT]
COMM_NO_COMMUNICATION
[CommunicationAllowed == FALSE]
FullCom requested [Mode Limitation is deactivated]
COMM_NO_COM_REQUEST_PENDING
COMM_NO_COM_NO_PENDING_REQUEST
Active Diagnostic Indication
Passive Wake-up indication
Pending FullCom request canceled
Inactive Diagnostic indication
ComM_Init()
Request NoCom
Start-up code or ComM_InitMemory()
COMM_UNINIT
Power Off

Figure 2-1 COMM channel state machine

The sub-states described below represent intermediate states, which perform activities to support a synchronized transition with external partners and managing protocols (e.g. Nm). The state machine is implemented for each communication channel independently.

COMM_UNINIT

Before the COMM is initialized it stays in this state. The COMM functionality cannot be used.

COMM_NO_COMMUNICATION

The state COMM_NO_COMMUNICATION represents the lowest state of the COMM. Inside this state no communication capability is available. The state consists of two sub-states described below.

COMM_NO_COM_NO_PENDING_REQUEST

This state is the default state after the initialization of the COMM.

The COMM resides in this state until the state COMM_FULL_COMMUNICATION is requested. There are different triggers to start communication:

> A user request COMM_FULL_COMMUNICATION and no Mode Inhibition is active or

> DCM notification of an active diagnostic session or

> A Passive Wake-up indication from ECUM or NM. If Synchronous Wake-up is enabled, the indication is propagated to all channels, which are not already in COMM_FULL_COMMUNICATION state.

If the requested COMM channel is a managed channel, then COMM also requests COMM_FULL_COMMUNICATION for the referenced managing channel.

A managing channel switches back to this default substate, only if its referenced managed channels have no COMM_FULL_COMMUNICATION requests anymore.

COMM_NO_COM_REQUEST_PENDING

The COMM resides in this state until the communication will be allowed on the channel with means of the ComM_CommunicationAllowed(TRUE) indication or the requests for COMM_FULL_COMMUNICATION are rejected, i.e. COMM user requests COMM_NO_COMMUNICATION or DCM indicates an inactive diagnostic session.

COMM_SILENT_COMMUNICATION

The COMM uses this state to support the sleep process of the network management. The state represents the prepare bus sleep phase of the network. The COMM changes into this state if the network management triggers the sleep process and changes into the prepare bus sleep mode.

Note

> Users cannot request this state directly.

> This state is available for Nm Variants FULL and PASSIVE with bus types CAN and Ethernet only. For other bus types, it is skipped.

> This state is available for Nm Variant LIGHT and bus type CAN if Nm Light Silent Timeout is configured.

> Note that Ethernet State Manager ignores requests for COMM_SILENT_COMMUNICATION mode, see [4]. COMM requests it for the sake of consistency when UDP Network Management indicates prepare bus sleep mode, see [5].

The COMM resides in this state until:

> The Network Management indicates a restart after receiving a NM message or

> The Network Management indicates bus sleep mode or

- A user requests COMM_FULL_COMMUNICATION again or
- DCM indicates an active diagnostic session.

COMM_FULL_COMMUNICATION

The state COMM_FULL_COMMUNICATION represents the highest state of the COMM. Inside this state the communication capability is available. The state consists of two sub-states described below.

COMM_FULL_COM_NETWORK_REQUESTED

The activity in this state depends on the configured COMM NM Variant:

- NM Variant FULL
  - The network management is set into the “Normal Operation” state.
  - COMM resides in this state until the following conditions are fulfilled:
    - All users request No Communication and DCM indicated no active diagnostic session.
    - The optional channel-specific Minimum Full Com Mode timer is expired, refer to chapter 2.1.2.4.
- NM Variant PASSIVE
  - COMM enters COMM_FULL_COM_READY_SLEEP state directly.
- NM Variant LIGHT
  - Case 1: transition from COMM_NO_COM_REQUEST_PENDING to COMM_FULL_COM_NETWORK_REQUESTED is triggered by a Passive Wake-up event. All users request No Communication and DCM indicates no active session. If the indicated channel is a managed channel, the COMM module will request a Passive Wake-up from the Network Management on the referencing managing channel.
    - COMM starts the “Minimum Full Communication Mode Timer”,
    - COMM resides in this state until “Minimum Full Communication Mode Time” is expired.
  - Case 2: a user requests Full Communication or DCM indicates an active diagnostic session
    - COMM cancels the “Minimum Full Communication Mode Timer” if the timer is started.
    - COMM resides in this state until all users request No Communication and DCM indicated no active diagnostic session. If the channel is a managing channel the state machine switches to sub-state COMM_FULL_COM_READY_SLEEP only if additional all managed channels have no user request.

- NM Variant NONE
  - COMM resides in this state. Shutdown of communication is done by an ECU reset or power off.
- NM Variant LINSLAVE
  - COMM resides in this state until all users request No Communication.
  - In case of a Passive Wake-Up the target state is COMM_FULL_COM_NETWORK_REQUESTED, if no users request full communication, the channel immediately transitions to COMM_FULL_COM_READY_SLEEP.

COMM_FULL_COM_READY_SLEEP

The activity inside this state depends on the configured COMM NM Variant:

- NM Variant FULL and PASSIVE
  - The network management is set into the Ready Sleep state.
  - COMM resides in this state until the NM triggers the sleep process or a user requests Full Communication again or DCM indicates an active diagnostic session.
- NM Variant LIGHT
  - COMM starts the Nm Light Timeout timer if the value configured is greater than 0s.
  - It resides in this state until the Nm Light Timeout timer expires, or a user requests Full Communication or DCM indicates an active diagnostic session.
  - If the optional Nm Light Silent Timeout is configured greater than 0s, COMM enters COMM_SILENT_COMMUNICATION. Otherwise the next state is COMM_NO_COM_NO_PENDING_REQUEST.
  - If Nm Light Timeout timer is configured to 0s, COMM omits the state and enters the next state directly.
- NM Variant NONE
  - This state is not available for this NM variant.
- NM Variant LINSLAVE
  - COMM resides in this state until bus sleep mode is indicated by LINSM. Shutdown synchronization is done by LIN master.

2.5 Partial Network Cluster

This chapter describes the Partial Network behavior, including Partial Network States and Gateway behavior, implemented by the Communication Manager.

2.5.1 Partial Network States

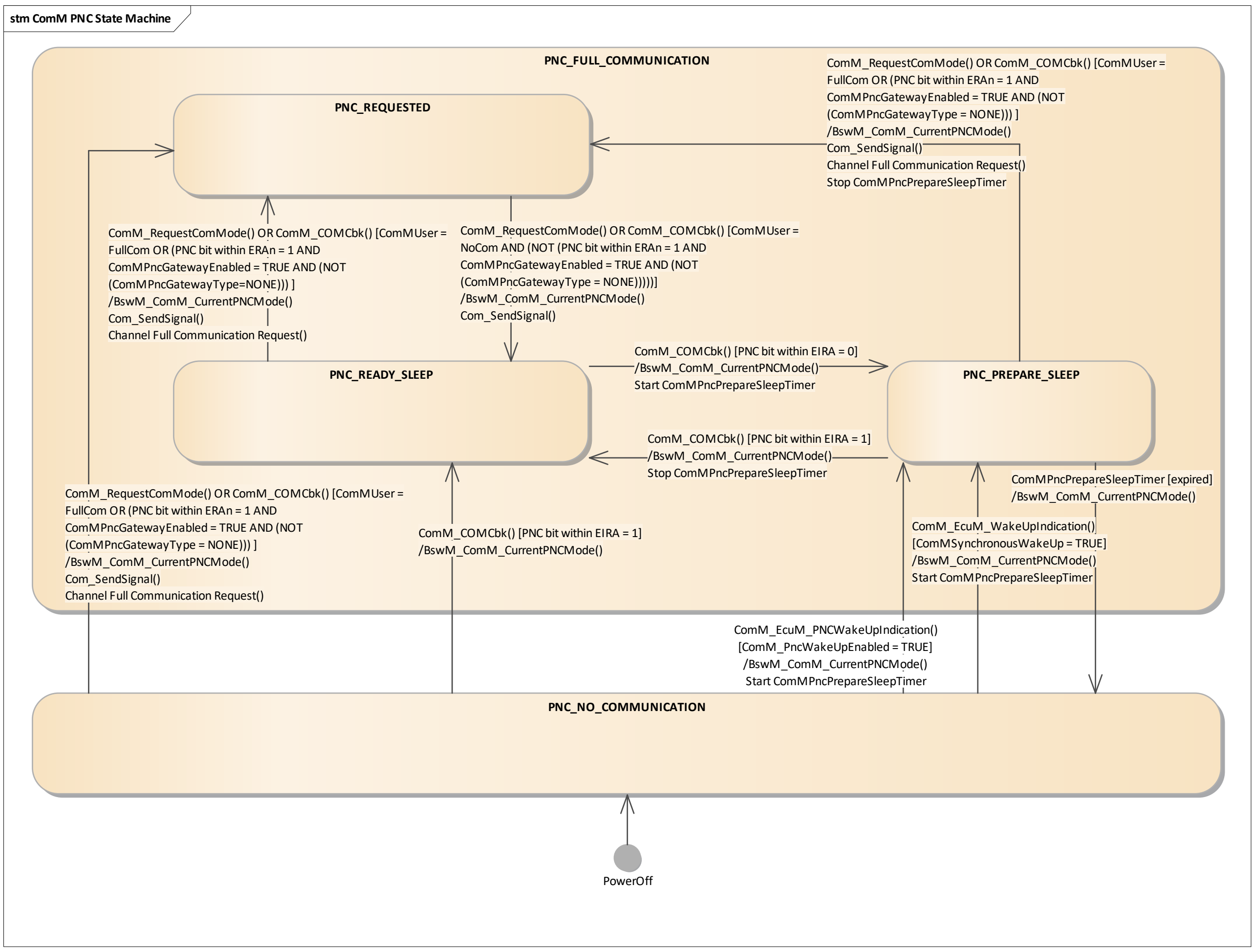
stm ComM PNC State Machine
PNC_FULL_COMMUNICATION
PNC_REQUESTED
ComM_RequestComMode() OR ComM_COMCbk() [ComMUser = FullCom OR (PNC bit within ERAn = 1 AND ComMPncGatewayEnabled = TRUE AND (NOT (ComMPncGatewayType = NONE)))] /BswM_ComM_CurrentPNCMode() Com_SendSignal() Channel Full Communication Request() Stop ComMPncPrepareSleepTimer
ComM_RequestComMode() OR ComM_COMCbk() [ComMUser = FullCom OR (PNC bit within ERAn = 1 AND ComMPncGatewayEnabled = TRUE AND (NOT (ComMPncGatewayType=NONE)))] /BswM_ComM_CurrentPNCMode() Com_SendSignal() Channel Full Communication Request()
ComM_RequestComMode() OR ComM_COMCbk() [ComMUser = NoCom AND (NOT (PNC bit within ERAn = 1 AND ComMPncGatewayEnabled = TRUE AND (NOT (ComMPncGatewayType = NONE)))))] /BswM_ComM_CurrentPNCMode() Com_SendSignal()
ComM_COMCbk() [PNC bit within EIRA = 0] /BswM_ComM_CurrentPNCMode() Start ComMPncPrepareSleepTimer
PNC_READY_SLEEP
PNC_PREPARE_SLEEP
ComM_COMCbk() [PNC bit within EIRA = 1] /BswM_ComM_CurrentPNCMode() Stop ComMPncPrepareSleepTimer
ComMPncPrepareSleepTimer [expired] /BswM_ComM_CurrentPNCMode()
ComM_RequestComMode() OR ComM_COMCbk() [ComMUser = FullCom OR (PNC bit within ERAn = 1 AND ComMPncGatewayEnabled = TRUE AND (NOT (ComMPncGatewayType = NONE)))] /BswM_ComM_CurrentPNCMode() Com_SendSignal() Channel Full Communication Request()
ComM_COMCbk() [PNC bit within EIRA = 1] /BswM_ComM_CurrentPNCMode()
ComM_EcuM_WakeUpIndication() [ComMSynchronousWakeUp = TRUE] /BswM_ComM_CurrentPNCMode() Start ComMPncPrepareSleepTimer
ComM_EcuM_PNCWakeUpIndication() [ComM_PncWakeUpEnabled = TRUE] /BswM_ComM_CurrentPNCMode() Start ComMPncPrepareSleepTimer
PNC_NO_COMMUNICATION
PowerOff

Figure 2-2 COMM Partial Network Cluster state machine

As shown in Figure 2-2 the COMM partial network state machine consists of two main states representing the abstract status of the partial network cluster (PNC). The state machine exists per partial network.

COMM uses two types of bit vector named EIRA and ERA to exchange PNC status information with other ECUs. Note that ERA is only evaluated if PNC Gateway feature is enabled and only for PNCs which are coordinated, i.e. assigned to more than one COMM channel. Each PNC uses the same bit position within the bit vectors, which is defined by the PNC ID. The status of a PNC within a bit vector (signal) can be

- **active** if the bit position of the PNC is 1 or
- **inactive** if the bit position of the PNC is 0.

PNC_NO_COMMUNICATION

This state is the default state after the initialization of the COMM.

The partial network leaves this state if one of the following events occurs:

> A user requests the state Full Communication for a partial network or

> EcuM or NM inform COMM about an external wake-up event and ComMPncPrepareSleepTimer is configured with a value > 0 and ‘Synchronous Wake-Up’ feature is enabled or

> COMM receives an EIRA or ERA signal which signalized the activation of the partial network.

PNC_FULL_COMMUNICATION

The state consists of three sub-states described below.

PNC_REQUESTED

The partial network reaches this state if one of the following events occurs:

> COMM user requests COMM_FULL_COMMUNICATION for this partial network or

> An ERA signal with partial network = active is received and the partial network is coordinated.

The state will be left if all COMM users for the corresponding partial network request COMM_NO_COMMUNICATION and the ERA signals for the corresponding partial network are received with status inactive.

PNC_READY_SLEEP

The partial network reaches this state if the following events occurred:

> All COMM users for the partial network request COMM_NO_COMMUNICATION and

> An EIRA signal is received with partial network = active and

> All ERA signals are received with partial network = inactive and the partial network is coordinated.

The state will be left if a COMM user for PNC requests COMM_FULL_COMMUNICATION or EIRA is received with partial network = inactive or ERA is received with partial network = active.

PNC_PREPARE_SLEEP

The partial network reaches this state if one of the following events occurs:

> An EIRA signal is received with partial network = inactive or

> EcuM notified a passive wake-up event and ‘Synchronous Wake-Up’ feature is enabled and ComMPncPrepareSleepTimer is configured with a value > 0.

The state will be left if a COMM user for PNC requests COMM_FULL_COMMUNICATION or EIRA/ERA signals are received with partial network = active or the ComMPncPrepareSleepTimer is expired.

If ComMPncPrepareSleepTimer is configured with 0, the state PNC_PREPARE_SLEEP is omitted when de-activating the partial network but it is still notified to BswM for the sake of completeness.

Caution
PNC Prepare Sleep Timer shall expire before Network Management leaves Ready Sleep state when shutting down the communication.

The crucial time is the time between indication of EIRA signal with PNC status = inactive and indication of Prepare Bus Sleep on CAN or Bus Sleep on FlexRay. The calculation of the exact time value depends on the bus type.

> On CAN:

CanNm Timeout Time (lowest of all CAN channels) – CanNm PNC Reset Time

> On FlexRay:

((Ready Sleep Count+1)*Repetition Cycle*<Duration of FlexRay Cycles>) – FrNm PNC Reset Time

> On Ethernet:

UdpNm Timeout Time (lowest of all Ethernet channels) – UdpNm PNC Reset Time

Calculate the lowest BusNm specific Timeout Time according to the rules given above. Then choose the PNC Prepare Sleep Timer to be less than (lowest BusNm specific Timeout Time) – (COMM Main Function Period of Channel with ID 0).

Caution
Only the COMM module is allowed to write the TX EIRA signals. The application must not write the TX EIRA signals by its own.

Note
COMM supports PNC mapping for managing channels and for managed channels (Nm Variant LIGHT). However, if a managed channel is referenced by a PNC, the corresponding managing channel must also be referenced by the same PNC. Managed channels, mapped to a PNC, always have GW type NONE and thus cannot be coordinated. The coordination and signal routing is only handled by the corresponding managing channel.

2.5.2 Partial Network Gateway and Routing Behavior

COMM is responsible for the gateway behavior of coordinated partial networks, i.e. the routing of PNC activation requests from one channel to other channels. The routing is done by sending of EIRA TX signals. The routing to a channel depends on the gateway type of this channel.

The basic behavior is as follows:

PNC activation request received on passive channel

If a request is received via ERA=1 on a channel with gateway type PASSIVE, the request is not mirrored back to the channel, i.e. the request is not set in the EIRA Tx signal, and it is not routed to other channels with gateway type PASSIVE. The request is only routed to channels with gateway type ACTIVE.

The target PNC state is COMM_PNC_REQUESTED. The target state for coordinated channels is dependent on the gateway type. For coordinated channels with gateway type PASSIVE the target state is COMM_FULL_COM_READY_SLEEP and the target state for coordinated channels with gateway type ACTIVE is COMM_FULL_COM_NETWORK_REQUESTED.

PNC activation request received on active channel

If a request is received via ERA=1 on a channel with gateway type ACTIVE, the request is mirrored back and routed to all other coordinated channels. The target PNC state is COMM_PNC_REQUESTED. The target state for both, the coordinated channels with gateway type PASSIVE and ACTIVE, is COMM_FULL_COM_NETWORK_REQUESTED.

Special use-case

If a PNC activation request is received on a channel, which is not mapped to the requested PNC, the request is not routed to channels with gateway type PASSIVE, but it will be forwarded to the channels with gateway type ACTIVE.

Thus, the target state of the coordinated passive channels is COMM_FULL_COM_READY_SLEEP in this scenario, if there are no further PNC requests received on this channel.

The general gateway behavior described above can additionally be tailored by the parameter settings of ‘PNC-0-Vector Avoidance’, ‘PNC Extended Functionality’ and ‘PNC to Channel Routing Limitation’.

PNC activation request received on channel with gateway type NONE

If a request is received via ERA=1 on a channel with gateway type NONE, the request is not stored in the internal COMM ERA signal, i.e. it is ignored. Thus, the request not mirrored back on the channel and is not routed to any other channels, i.e. the request is not set in the EIRA Tx signal. The channels with gateway type NONE ignore the PNC requests received via ERA signal, but they process PNC requests received via EIRA Rx signal. In this case, the target PNC state is not affected by the request received via ERA, but it changes due to PNC requests received via EIRA=1. If a managed channel, of gateway type NONE, is referenced by a PNC, neither EIRA nor ERA signals are allowed for that channel.

Note
It must be ensured by the System/Application, that all referenced COMM Channels with GW type NONE are coordinated (requested and released at the same point in time). COMM does not wake up a PNC if any referenced channel is still not in FULL_COM.

PNC-0-Vector Avoidance

If the parameter ComM0PncVectorAvoidance is activated, the basic behavior described above is achieved, i.e. in case of a PNC request received on a passive channel the target state of the passive channel is COMM_FULL_COM_READY_SLEEP. Thus, it is avoided to send PNC-Vectors containing only ‘0’ on this channel. Deactivating this parameter changes the target state of the passive channels to COMM_FULL_COM_NETWORK_REQUESTED. Thus, the passive channel is not released and continues sending PNC-Vectors containing only ‘0’.

PNC Extended Functionality

If the optional parameter 'PNC Extended Functionality Enabled' is activated, additionally the following cases are supported:

- Coordinated PNC with an ERA signal and an additional channel
- PNC without channels, with an EIRA TX signal
- Coordinated PNC without channels, with ERA and EIRA TX signals
- Coordinated PNC mapped to one channel, with ERA and EIRA TX signals

PNC to Channel Routing Limitation

If the ‘PNC to Channel Routing Limitation’ is activated, a channel can be limited for a specific PNC (using the provided API ComM_LimitPncToChannelRouting()). This means, the limited channel will not be woken up by PNC activation requests received on another channel as long as the channel is limited. Thus, the gateway is not routing PNC requests on the limited channel (see also 2.1.2.5).

2.5.3 Synchronized PNC Shutdown

If the ‘Synchronized PNC Shutdown’ is activated a nearly synchronized PNC shutdown across the PN topology from the Top-Level PNC coordinator down to the subordinated PNC node is supported.

As an intermediate PNC coordinator, COMM only forwards a synchronized PNC shutdown request towards the NM if an indication to forward a synchronized PNC shutdown request has been previously received from a Top-Level PNC coordinator and if the affected PNC is released (i.e. PNC enters the state COMM_PNC_READY_SLEEP).

As a Top-Level PNC coordinator, COMM requests the synchronized PNC shutdown from NM as soon as the affected PNC is released (i.e. PNC enters the state COMM_PNC_READY_SLEEP).

Caution

> A request for a PNC, either internally or externally, always overrule a request for a synchronized PNC shutdown.

> This feature is not available for subordinated (or leaf) nodes.

2.6 Main Functions

This chapter describes how the Communication Manager features are to be used by upper software layers or application software and shows the interaction with other modules.

2.6.1 Communication Control Handling

The communication control handling is the main functionality of the communication manager. This functionality contains the following parts:

- Collection of the network wake-up events, means bus wake-up, user and DCM communication requests
- Verification of the network wake-up events and start of the corresponding network with regarding of the used NM variant.

The COMM supports the following NM variants:

- **FULL**, AUTOSAR NM is used. Also it is designed to support LIN NM and J1939 NM.
- **PASSIVE**, AUTOSAR NM is used, but the ECU is not allowed to keep the network awake. COMM ignores communication requests from users and DCM on this channel; refer to chapters 2.1.2, 4.4.6 and 4.4.7. The parameter ComMNoCom has to be set to true.
- **LIGHT**, no AUTOSAR NM is used, but the shutdown is synchronized via timeout, which is configured with the parameter ComMNmLightTimeout.
- **NONE**, no AUTOSAR NM is used and no shutdown synchronization is available. I.e. once a channel reached the COMM_FULL_COMMUNICATION mode, it will never leave it. Stop of communication is done via power off or reset. COMM ignores user requests of COMM_NO_COMMUNICATION mode and requests for No Communication Mode Limitation; refer to chapters 4.2.14 and 4.2.15.
- **LINSLAVE**, no AUTOSAR NM is used but the LIN State Manager restarts wakeup repetition. Communication shutdown is synchronized by LIN master.

Caution
If LIN NM is used on a COMM channel, its NM Variant shall be 'FULL'.

LIN NM does not trigger communication shutdown after COMM called `Nm_PassiveStartUp`. This can prevent the ECU from entering the sleep state. To avoid this, apply one of the workarounds described in the Technical Reference of MICROSAR LIN Network Management [6].

Caution
If J1939 NM is used on a COMM channel, its NM Variant shall be 'FULL'.

J1939 NM does not provide `Nm_PassiveStartUp` API. Therefore channels with J1939 NM cannot be woken up externally. Ensure that parameter 'Synchronous Wake Up' is disabled in the COMM module configuration (see chapter 2.6.3).

COMM_BUS_TYPE_INTERNAL shall be configured if a channel is used for internal communication only. Such channels have no corresponding bus interface. Only NM Variant LIGHT is supported for channels with COMM_BUS_TYPE_INTERNAL.

Caution
Only NM Variant FULL is supported for the managing channels.

Since the managing channel is responsible for the interaction with NM, only NM Variant LIGHT is supported for managed channels.

2.6.2 Mode Limitation

Mode limitation is a mechanism to restrict the actions of the COMM user, especially the requesting of communication modes. The COMM supports 2 different mode limitation mechanisms:

> No Communication mode limitation and

> Prevent Wake-up.

The mode limitation mechanism can be used to restrict the communication requests of ECUs which wrongly keep the bus awake.

2.6.2.1 Mode Limitation to NO_COM

This mechanism can be used to force COMM channel(s) into the sleep mode although one or more COMM user requests COMM_FULL_COMMUNICATION. Note that this is not supported for Nm Variant NONE.

The limitation can be activated/deactivated via ComM_LimitChannelToNoComMode(), for a specific channel, or via ComM_LimitECUToNoComMode() for the whole ECU.

If Mode Limitation is active, COMM ignores new COMM_FULL_COMMUNICATION mode requests and triggers communication shutdown on the channel. When ComM_LimitChannelToNoComMode is called, COMM updates the inhibition status (limitation to NoCom) for the corresponding channel. An update of the inhibition status due to a request for limit to COMM_NO_COMMUNICATION is always performed, independent of the current state but is ignored until the channel gets requested.

When a channel switches to NO_COM mode due to an active Mode Limitation, COMM clears all COMM_FULL_COMMUNICATION requests of users that are mapped to the channel directly or via PNC. New COMM_FULL_COMMUNICATION mode requests are stored but not performed as long as Mode Limitation is active. The requests are performed if Mode Limitation is deactivated.

Reset after Forcing NO_COM functionality

When a channel switches to NO_COM mode due to an active Mode Limitation (i.e. BusSM indicates NO_COM), COMM requests an ECU reset by calling BswM_ComM_InitiateReset if the following conditions are fulfilled:

1. Reset after Forcing NO_COM functionality is enabled and
2. BusSM indicated NO_COM for all channels and
3. All channels are in NO_COM mode. Possible bus wake-ups are ignored in order to trigger a reset as soon as possible.

Notes:

> The purpose of conditions 2 and 3 is to ensure a controlled reset, i.e. to avoid that a reset is performed during active bus communication.

> Conditions 2 and 3 are not applicable for channels with Nm Variant NONE.

2.6.2.2 Prevent Wake-Up

Prevent Wake-Up is the second mode limitation mechanism; it avoids that COMM channels can be woken up via a COMM_FULL_COMMUNICATION request by a COMM user. Prevent wake-up can be activated/deactivated via ComM_PreventWakeUp() but the limitation is only performed if the current state of the COMM channel is COMM_NO_COMMUNICATION or COMM_SILENT_COMMUNICATION. User requests for COMM_FULL_COMMUNICATION are stored but not performed. The requests are performed if Prevent Wake-up is deactivated.

Note
The prevent wake-up state is stored in the non-volatile memory.

2.6.3 Synchronous Wake-Up

Caution
Synchronous wake-up does only trigger the wake-up but is not responsible to keep all busses awake or responsible for a synchronous shutdown of these busses.
Synchronous shutdown of multiple channels is the responsibility of Nm coordinator.

If **J1939** NM is used on a channel, Synchronous wake-up must be disabled in the COMM module configuration, consider the notes in chapter 2.6.1.

Synchronous wake-up means, that the COMM triggers a wake-up of all COMM busses as soon as one COMM bus notifies an external wake–up, e.g. via ECUM wake-up notification or via the notification of the configured NM.

2.7 Error Handling

2.7.1 Development Error Reporting

Development errors are reported to the DET using the service `Det_ReportError()` as specified in [2], if development error reporting is enabled (i.e. pre-compile parameter `COMM_DEV_ERROR_DETECT==STD_ON`).

The reported COMM ID is 12 decimal.

The reported service IDs identify the services which are described in Table 2-5. The following table presents the service IDs and the related services:

Service ID	Service
0x01	ComM_Init
0x02	ComM_DeInit
0x03	ComM_GetStatus
0x04	ComM_GetInhibitionStatus
0x05	ComM_RequestComMode
0x06	ComM_GetMaxComMode
0x07	ComM_GetRequestedComMode
0x08	ComM_GetCurrentComMode
0x09	ComM_PreventWakeUp
0x0b	ComM_LimitChannelToNoComMode
0x0c	ComM_LimitECUToNoComMode
0x0d	ComM_ReadInhibitCounter
0x0e	ComM_ResetInhibitCounter
0x0f	ComM_SetECUGroupClassification
0x10	ComM_GetVersionInfo
0x15	ComM_Nm_NetworkStartIndication
0x18	ComM_Nm_NetworkMode
0x19	ComM_Nm_PrepareBusSleepMode
0x1a	ComM_Nm_BusSleepMode
0x1b	ComM_Nm_RestartIndication
0x1f	ComM_DCM_ActiveDiagnostic
0x20	ComM_DCM_InactiveDiagnostic
0x2a	ComM_EcuM_WakeUpIndication
0x2b	ComM_EcuM_PNCWakeUpIndication
0x33	ComM_BusSM_ModeIndication
0x34	ComM_BusSM_BusSleepMode
0x35	ComM_CommunicationAllowed

Service ID	Service
0x36	ComM_LimitPncToChannelRouting
0x60	ComM_MainFunction
0x37	ComM_GetDcmRequestStatus
0x38	ComM_GetMinFullComModeTimerStatus
0x39	ComM_GetState
0x61	ComM_MainFunction_Satellite()
0x62	ComM_PreInit()
0x63	ComM_PostInit()
0x6a	ComM_GetCurrentPNCComMode
0x6b	ComM_Nm_ForwardSynchronizedPncShutdown

Table 2-5 Service IDs

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

Error Code		Description
0x01	COMM_E_NOT_INITED	API service used without module initialization
0x02	COMM_E_WRONG_PARAMETERS	API service used with wrong parameters
0x03	COMM_E_ERROR_IN_PROV_SERVICE	Provided API service of other modules returned with an error
0x04	COMM_E_NOSUPPORTED_MODECHANGE	COMM tries to perform a not allowed state change
0x05	COMM_E_DIAGNOSTIC_NOT_SUPPORTED	Diagnostic communication is requested or released on a channel where diagnostic is not supported. This happens when Nm Variant PASSIVE or LINSLAVE is configured on the channel.
0x06	COMM_E_ALREADY_INITIALIZED	The service ComM_Init is called while the module is already initialized
0x07	COMM_E_WRONG_GENERATED_DATA	Check of generated data has failed
0x08	COMM_E_NO_PREINIT	API service is used without module pre-initialization
0x09	COMM_E_INVALID_PARTITION	The current partition index is out of range
0x10	COMM_E_NO_POSTINIT	API service is used without module post-initialization

Table 2-6 Errors reported to DET

2.7.1.1 Parameter Checking

AUTOSAR requires that API functions check the validity of their parameters. The checks in Table 2-7 are internal parameter checks of the API functions. These checks are for development error reporting and can be enabled or disabled via the parameter `COMM_DEV_ERROR_DETECT`.

The following table shows which parameter checks are performed on which services:

Check Service	COMM_E_NOT_INITED	COMM_E_WRONG_PARAMETERS	COMM_E_ERROR_IN_PROV_SERVICE	COMM_E_NOTSUPPORTED_MODECHANGE	COMM_E_DIAGNOSTIC_NOT_SUPPORTED	CCOM_E_INVALID_PARTITION	COMM_E_ALREADY_INITIALIZED	COMM_E_WRONG_GENERATED_DATA	COMM_E_NO_PREINIT	COMM_E_NO_POSTINIT
ComM_Init		■	■			■	■		■	
ComM_DeInit			■			■				
ComM_GetStatus		■								
ComM_GetState		■				■				■
ComM_GetInhibitionStatus		■								■
ComM_RequestComMode		■								■
ComM_GetMaxComMode		■								■
ComM_GetRequestedComMode		■								■
ComM_GetCurrentComMode		■	■							■
ComM_GetCurrentPNCComMode		■								■
ComM_PreventWakeUp		■				■				■
ComM_LimitChannelToNoComMode		■				■				■
ComM_LimitECUToNoComMode						■				■
ComM_ReadInhibitCounter		■				■				■
ComM_ResetInhibitCounter										■
ComM_SetECUGroupClassification		■				■				■
ComM_GetVersionInfo		■								
ComM_MainFunction		■	■	■						■
ComM_CommunicationAllowed		■								■
ComM_Nm_NetworkStartIndication		■								■
ComM_Nm_NetworkMode		■								■
ComM_Nm_PrepareBusSleepMode		■								■
ComM_Nm_BusSleepMode		■								■
ComM_Nm_RestartIndication		■								■

Check Service	COMM_E_NOT_INITED	COMM_E_WRONG_PARAMETERS	COMM_E_ERROR_IN_PROV_SERVICE	COMM_E_NOTSUPPORTED_MODECHANGE	COMM_E_DIAGNOSTIC_NOT_SUPPORTED	CCOM_E_INVALID_PARTITION	COMM_E_ALREADY_INITIALIZED	COMM_E_WRONG_GENERATED_DATA	COMM_E_NO_PREINIT	COMM_E_NO_POSTINIT
ComM_DCM_ActiveDiagnostic		■			■					■
ComM_DCM_InactiveDiagnostic		■			■					■
ComM_EcuM_WakeUpIndication		■								■
ComM_EcuM_PNCWakeUpIndication		■								■
ComM_BusSM_ModeIndication		■								■
ComM_BusSM_BusSleepMode		■			■					■
ComM_TF_NoCom_NetReq			■							
ComM_TF_ReadyS_NetReq			■							
ComM_TF_NetReq_ReadyS			■							
ComM_PreInit		■								
ComM_PostInit	■									
ComM_MainFunction_Satellite						■				■
ComM_InitCheckGeneratedData								■		
ComM_LimitPncToChannelRouting		■				■		■		
ComM_GetDcmRequestStatus		■				■				■
ComM_MainFunction_Satellite										
ComM_TF_NoCom_FullReadySleep			■							
ComM_TF_Full_SiCom			■							
ComM_TF_SiCom_NoCom			■							
ComM_ForwardRequestBusSmMode			■							
ComM_TF_No_Transition				■						
ComM_RequestComMode		■				■				■
ComM_GetMinFullComModeTimerStatus		■								■
ComM_Nm_StateChangeNotification		■								■
ComM_ Nm_ForwardSynchronizedPncShutdown		■								■

Table 2-7 Development Error Reporting: Assignment of checks to services

2.7.2 Production Code Error Reporting

COMM does not report any production errors.

3 Integration

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

3.1 Scope of Delivery

The delivery of the COMM contains the files which are described in the following chapters.

3.1.1 Static Files

File Name	Source Code Delivery	Object Code Delivery	Description
ComM.c	■		This is the source file of the COMM. It contains the implementation of the main functionality.
ComM.h	■		This is the header file of the COMM, which is the interface for upper layers to the services of the COMM.
ComM_Types.h	■		Header File which includes COMM specific data types.
ComM_BusSM.h	■		Header File which includes the external declarations of the Bus State Manager callback functions.
ComM_EcuMBswM.h	■		Header File which includes the external declarations of the EcuM and BswM callback functions.
ComM_Nm.h	■		Header File which includes the external declarations of the Nm callback functions.
ComM_Dcm.h	■		Header File which includes the external declarations of the Dcm callback functions.

Table 3-1 Static files

3.1.2 Dynamic Files

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

File Name	Description
ComM_Lcfg.c	This is the link time configuration source file. It contains all link time configuration settings.
ComM_Lcfg.h	This is the link time configuration header file.
ComM_Cfg.h	This is the COMM configuration header file.
ComM_GenTypes.h	This file contains the generated type definitions of the COMM.
ComM_PBcfg.c	Post-build variant configuration source file.
ComM_Private_Cfg.h	This file contains generated types, macros, and #includes which are needed by COMM implementation but not exposed through ComM.h

File Name	Description
ComM_MemMap.h	This is the header file containing the ComM specific macros for memory mapping.

Table 3-2 Generated files

In case of Multi-Partition additional partition specific files are generated.

File Name	Description
ComM_Lcfg_<OsApplication>.c	This is the link time configuration source file for the respective OsApplication. It contains all link time configuration settings.
ComM_Lcfg_<OsApplication>.h	This is the link time configuration header file for the respective OsApplication.
ComM_PBcfg_<OsApplication>.c	This is the post build time configuration source file for the respective OsApplication. It contains all the post build configuration settings.
ComM_PBcfg_(OsApplication>.h	This is the post build time configuration header file for the respective OsApplication.

Table 3-3 Generated Multi-Partition files

3.2 Include Structure

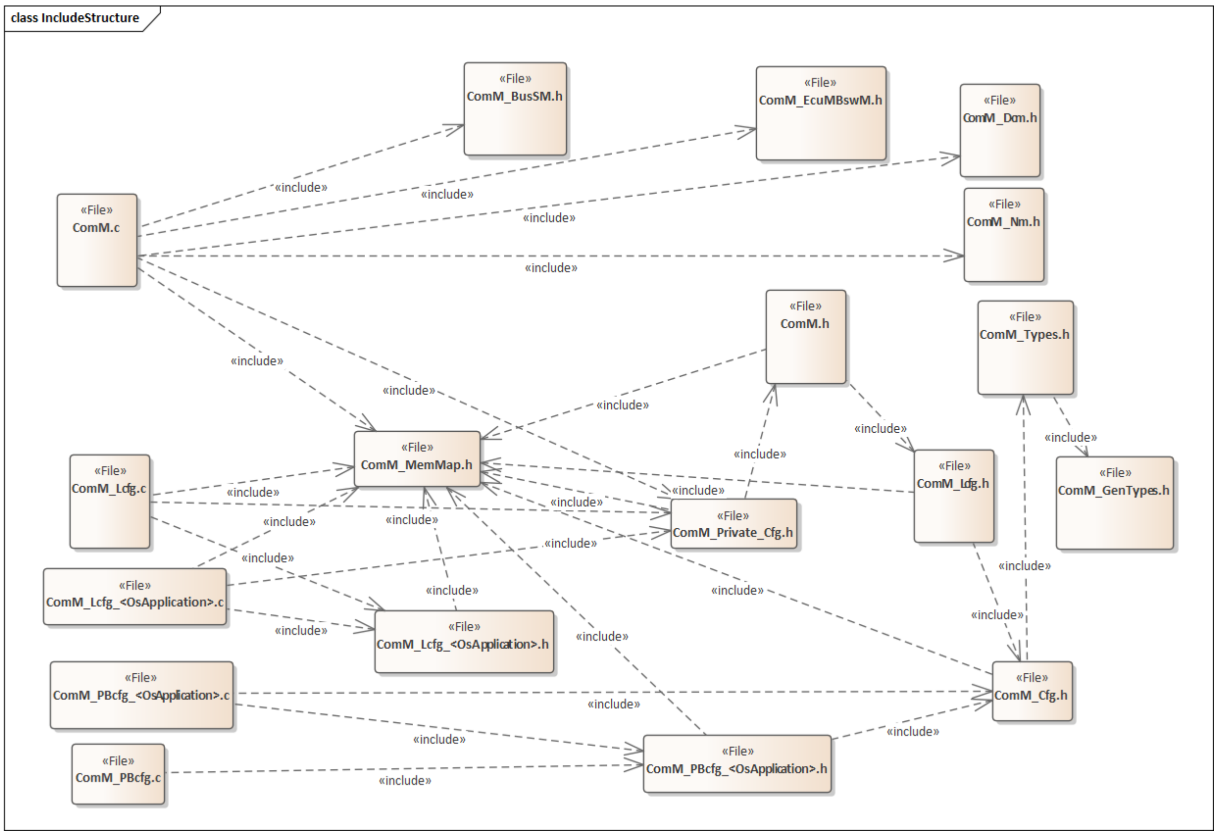

Figure 3-1 Include structure

3.3 Critical Sections

COMM requires the following critical code sections:

COMM_EXCLUSIVE_AREA_0

- Configuration: This critical section must lock task interrupts and interrupt sources.
- Purpose: This critical section protects the channel state and user request status.
- This critical section covers calls to several sub-functions and can have a long run-time.

COMM_EXCLUSIVE_AREA_1

- Configuration: This critical section must lock task interrupts if ComM_MainFunction() can be interrupted by one of the following BSW Module tasks. Otherwise no interrupt lock is necessary.
  - Nm_MainFunction()
  - BusNm_MainFunction(), e.g. CanNm_MainFunction()
  - BusSM_MainFunction(), e.g. CanSM_MainFunction()
  - If 'Pnc Support' is enabled in the module configuration, it must be ensured that ComM_MainFunction() is not interrupted by ComM_RequestComMode(). If an interruption is possible, the section requires global interrupt lock.
- Purpose: This critical section protects the channel state.
- This critical section covers calls to several sub-functions and can have a long run-time.

For Multi-Partition configurations, additionally the following critical sections have to be configured:

COMM_EXCLUSIVE_AREA_2

- Configuration: This critical section must be configured as a spin lock to protect the consistency of synchronized data. It must only be configured for Multi-Partition configurations.
- Purpose: This critical section protects the synchronization of the channel state variables.
- This critical section covers calls to several sub-functions and can have a long run-time.

COMM_EXCLUSIVE_AREA_3

- Configuration: This critical section must be configured as a spin lock to protect the consistency of synchronized data. It must only be configured for Multi-Partition configurations.
- Purpose: This critical section protects the synchronization of PNC requests received via ERA.
- This critical section covers calls to several sub-functions and can have a long run-time.

COMM_EXCLUSIVE_AREA_4

> Configuration: This critical section must be configured as a spin lock to protect the consistency of synchronized data. It must only be configured for Multi-Partition configurations.

> Purpose: This critical section protects the synchronization of DCM requests and state changes due to BusSM indications.

COMM_EXCLUSIVE_AREA_5

> Configuration: This critical section must be configured as a spin lock to protect the consistency of synchronized data. It must only be configured for Multi-Partition configurations.

> Purpose: This critical section protects the synchronization of PNC signal values and PNC to Channel Routing states.

COMM_EXCLUSIVE_AREA_6

> Configuration: This critical section must be configured as a spin lock to protect the consistency of synchronized data. It must only be configured for Multi-Partition configurations.

> Purpose: This critical section protects the synchronization of external PNC requests.

COMM_EXCLUSIVE_AREA_7

> Configuration: This critical section must be configured as a spin lock to protect the consistency of synchronized data. It must only be configured for Multi-Partition configurations.

> Purpose: This critical section protects the synchronization of forwarding the BusSM requests and COM Signal updates to the respective partition.

Note
It is recommended to use OS Resources for the exclusive areas COMM_EXCLUSIVE_AREA_0 and COMM_EXCLUSIVE_AREA_1 to prevent priority inversions and deadlocks.

Using OS Resources for COMM_EXCLUSIVE_AREA_0 is not possible if 'Pnc Support' is enabled in the module configuration.

Using OS Resources for COMM_EXCLUSIVE_AREA_1 is not possible if 'Multi-Partition Support' is enabled in the module configuration. In this case, using OS Interrupt Blocking is recommended.

3.4 Handling of non-volatile Data

The non-volatile data is handled via the AUTOSAR NvRAM Manager. The COMM uses the following NvRAM Manager API:

- `NvM_GetErrorStatus(..)` The non-volatile data must be loaded and stored in the below listed variable before the COMM is initialized via ComM_Init(). The COMM checks with the function NvM_GetErrorStatus(..) if the COMM data is loaded or not. If not then the COMM works with the configured values of the ECU Group Classification and prevent wake-up state. Additionally the COMM resets the inhibition counter to 0.
- `NvM_SetRamBlockStatus(..)` This function is used to trigger the storage of the non-volatile data.

The non-volatile data of the COMM are grouped inside the structure called `ComM_Inhibition`. The structure contains the following elements (order of elements equal to the structure element order):

- `ComM_ECUGroupClassification`
  - size: 1 Byte
  - stores the ECU Group classification
- `ComM_InhibitCnt`
  - size: 2 Byte
  - stores the inhibition counter
- `ComM_InhibitionStatus[<COMM_ACTIVE_CHANNEL>]`
  - size: 1 Byte per COMM channel
  - stores the prevent wake up state

Caution
The COMM non-volatile data must be loaded and stored inside the above listed variables before the COMM is initialized via `ComM_Init()`. If not, COMM will use the configured values.

The non-volatile data handling is only necessary if at least one of the COMM configuration options Mode Limitation or Wake-up Inhibition is enabled.

3.5 Multi-Partition

For configurations with Multi-Partition support it has to be ensured that all API calls originating from the Application (e.g. ComM_RequestComMode()) and BswM (e.g. ComM_CommunicationAllowed) towards ComM are performed on the ComM master partition. The ComM master partition will distribute the requests to the corresponding satellite partition, if needed.

Note

The ComM provides only one SWC instance which is assigned to the master partition. This ensures that the RTE can route all requests from the application to the master partition.

After initialization, it has to be ensured that the ComM_ConfigPtr is protected against memory write accesses from partitions with a lower ASIL than the ComM master partition. The ComM_ConfigPtr is mapped to the memory section COMM_START_SEC_VAR_ZERO_INIT_UNSPECIFIED.

4 API Description

For an interfaces overview please see Figure 1-2.

4.1 Type Definitions

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

Type Name	C-Type	Description	Value Range
ComM_InitStatusType	uint8	Initialization status of COMM.	`COMM_UNINIT` COMM is not initialized
			`COMM_INIT` COMM is initialized and usable
ComM_InhibitionStatusType	uint8	Inhibition status of COMM	Bit 0 (LSB): 0 - Wake-up Inhibition is not active 1 - Wake-up Inhibition is active
			Bit 1: 0 - Mode Limitation is not active 1 - Mode Limitation is active
ComM_UserHandleType	uint16	Handle to identify a COMM user	0..65535 Note: ID 65535 is reserved
ComM_BusType	uint8	Configured Bus Type of a COMM Channel	`COMM_BUS_TYPE_CAN` The channel is a CAN Channel
			`COMM_BUS_TYPE_FR` The channel is a FlexRay channel
			`COMM_BUS_TYPE_LIN` The channel is a LIN channel
			`COMM_BUS_TYPE_ETH` The channel is an Ethernet channel
			`COMM_BUS_TYPE_INTERNAL` The channel is an INTERNAL channel
ComM_ModeType	uint8	Current COMM mode (main state of the state machine)	`COMM_NO_COMMUNICATION` COMM is in the state No Communication
			`COMM_SILENT_COMMUNICATION` COMM is in state Silent Communication
			`COMM_FULL_COMMUNICATION` COMM is in state Full Communication
ComM_PncModeType	uint8	Current mode of a PNC	`COMM_PNC_NO_COMMUNICATION` PNC is in the state No Communication
			`COMM_PNC_PREPARE_SLEEP` PNC is in state Prepare Sleep
			`COMM_PNC_READY_SLEEP` PNC is in state Ready Sleep

Type Name	C-Type	Description	Value Range
			`COMM_PNC_REQUESTED` PNC is in state Requested
ComM_StateType	uint8	State and sub-state of COMM state machine	`COMM_NO_COM_NO_PENDING_REQUEST`
			`COMM_NO_COM_REQUEST_PENDING`
			`COMM_FULL_COM_NETWORK_REQUESTED`
			`COMM_FULL_COM_READY_SLEEP`
			`COMM_SILENT_COM`
ComM_ConfigType	struct	Post-build configuration structure	-

Table 4-1 Type definitions

ComM_InhibitionType

This structure contains current inhibition status. It is stored non-volatile.

Struct Element Name	C-Type	Description	Value Range
ComM_ECUGroupClassification	uint8	Current ECU group classification	`0` ECU is not affected by mode inhibition
			`1` ECU is affected by Wake-up Inhibition only
			`2` ECU is affected by Mode Limitation only
			`3` ECU is affected by both inhibition types
ComM_InhibitCnt	uint16	Inhibition counter	`0..65535`
ComM_InhibitionStatus	uint8[]	Inhibition status per COMM channel	`0..3` Refer to description of ComM_InhibitionStatusType

Table 4-2 ComM_InhibitionType

ComM_UserHandleArrayType

This structure contains the set of COMM users requesting Full Communication for a channel.

Struct Element Name	C-Type	Description	Value Range
numberOfRequesters	uint16	Number of valid user handles in the handleArray member. The value is zero if no user keeps the channel requested.	`0..65534`
handleArray	ComM_UserHandleType[]	User handles of the users which keep the channel requested (if any), starting in its first entries. The size of the array is the number of users configured on the channel.	

Table 4-3 ComM_UserHandleArrayType

4.2 Services provided by COMM

4.2.1 ComM_InitMemory

Prototype	
`void` **`ComM_InitMemory`** `( void )`	
Parameter	
–	-
Return code	
–	-
Functional Description	
If RAM is not automatically initialized at start-up, this function must be called from start-up code to ensure that variables which must be initialized with a certain value (e.g. initialization status with `COMM_UNINIT` value) are set to those values.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant. > This function is a Vector Extension.	
Expected Caller Context	
> This function has to be called once during start-up and before `ComM_Init()` is called.	

Table 4-4 ComM_InitMemory

4.2.2 ComM_PreInit

Prototype	
`void` **`ComM_PreInit`** `( ComM_ConfigType* ConfigPtr )`	
Parameter	
`ConfigPtr`	Configuration pointer is needed if MICROSAR Identity Manager Post-Build Selectable or Post-Build Loadable is used.

Return code	
–	-
Functional Description	
This function initializes the ComM configuration pointer and pre-initializes the ComM.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant. > This function has to be called prior to the initialization of the individual ComM instances (i.e. > partitions).	
Expected Caller Context	
> This function can be called from task level only.	

Table 4-5 ComM_PreInit

4.2.3 ComM_Init

Prototype	
`void` **`ComM_Init`** `( ComM_ConfigType* ConfigPtr )`	
Parameter	
`ConfigPtr`	Configuration pointer is needed if MICROSAR Identity Manager Post-Build Selectable or Post-Build Loadable is used. Otherwise the function has no parameter.
Return code	
–	-
Functional Description	
This function initializes the COMM. All variables are set to default values. The COMM initialization state is set to `COMM_INIT` and the COMM main state is set to `COMM_NO_COM_NO_PENDING_REQUEST`.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant. > If Mode Limitation or Wake-up Inhibition is enabled, the non-volatile values must be loaded and stored before this function is called (refer to chapter ‘Handling of non-volatile Data’). > This function has to be called after the pre-initialization of the ComM.	
Expected Caller Context	
> This function can be called from task level only.	

Table 4-6 ComM_Init

4.2.4 ComM_PostInit

Prototype	
`void ComM_PostInit ( void )`	
Parameter	
–	-
Return code	
–	-
Functional Description	
This function finalizes the initialization of ComM.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant. > This function has to be called on the ComM Master Partition. > This function has to be called after the initialization of all individual ComM instances (i.e. > partitions).	
Expected Caller Context	
> This function can be called from task level only.	

Table 4-7 ComM_PostInit

4.2.5 ComM_DeInit

Prototype	
`void` **`ComM_DeInit`** `( void )`	
Parameter	
–	-
Return code	
–	-
Functional Description	
This function de-initializes COMM and sets the initialization status to `COMM_UNINIT`. It stores non-volatile values in NVRAM (refer to chapter 'Handling of non-volatile Data').	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant. > This function is executed if all COMM channels are in state `COMM_NO_COM_NO_PENDING_REQUEST`. Otherwise calling the function has no effect.	
Expected Caller Context	
> Function can be called in task and interrupt context	

Table 4-8 ComM_DeInit

4.2.6 ComM_GetStatus

Prototype	
`Std_ReturnType` **`ComM_GetStatus`** `( ComM_InitStatusType* Status )`	
Parameter	
`Status`	Pointer where the COMM initialization status shall be stored
Return code	
`E_OK`	ComM_GetStatus has performed
`E_NOT_OK`	Invalid parameter
Functional Description	
This function gets the initialization status of the COMM.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant.	
Expected Caller Context	
> Function can be called in task and interrupt context	

Table 4-9 ComM_GetStatus

4.2.7 ComM_GetInhibitionStatus

Prototype	
`Std_ReturnType` **`ComM_GetInhibitionStatus`** `( NetworkHandleType Channel, ComM_InihibitionStatusType* Status )`	
Parameter	
`Channel`	Index of the system channel
`Status`	Pointer where the COMM inhibition status shall be stored
Return code	
`E_OK`	Successfully returned Inhibition Status
`E_NOT_OK`	Invalid parameter
`COMM_E_UNINIT`	COMM is not initialized
Functional Description	
This function gets the current COMM inhibition status of the given channel.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant.	
Expected Caller Context	
> Function can be called in task and interrupt context	

Table 4-10 ComM_GetInhibitionStatus

4.2.8 ComM_RequestComMode

Prototype	
`Std_ReturnType` **`ComM_RequestComMode`** `( ComM_UserHandleType User, ComM_ModeType ComMode )`	
Parameter	
`User`	Index of the User, the user handles are generated and can be found in the ComM_Cfg.h file

`ComMode`	The requested communication mode:
	`COMM_FULL_COMMUNICATION`
	`COMM_NO_COMMUNICATION`

Return code	
`E_OK`	Request is accepted
`E_NOT_OK`	Invalid parameter
`COMM_E_UNINIT`	COMM is not initialized
`COMM_E_MODE_LIMITATION`	Requested was successful but mode cannot be granted because of mode inhibition

Functional Description
This function is used by upper layer modules or application to request the given communication mode. The communication mode request is stored and will be processed in the `ComM_MainFunction()`.
Particularities and Limitations
> Service ID: see table 'Service IDs' > This function is asynchronous. > This function is non-reentrant.
Expected Caller Context
> Function can be called in task and interrupt context

Table 4-11 ComM_RequestComMode

4.2.9 ComM_GetMaxComMode

Prototype	
`Std_ReturnType` **`ComM_GetMaxComMode`** `( ComM_UserHandleType User, ComM_ModeType* ComMode )`	
Parameter	
`User`	Index of the User, the user handles are generated and can be found in the ComM_Cfg.h file
`ComMode`	Pointer where the maximal communication mode of the given user shall be stored
Return code	
`E_OK`	Request is accepted
`E_NOT_OK`	Invalid parameter
`COMM_E_UNINIT`	COMM is not initialized
Functional Description	
This function queries the maximum allowed communication mode of the corresponding user.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant.	
Expected Caller Context	
> Function can be called in task and interrupt context	

Table 4-12 ComM_GetMaxComMode

4.2.10 ComM_GetRequestedComMode

Prototype
`Std_ReturnType` **`ComM_GetRequestedComMode`** `( ComM_UserHandleType User, ComM_ModeType* ComMode )`

Parameter	
`User`	Index of the User, the user handles are generated and can be found in the ComM_Cfg.h file
`ComMode`	Pointer where the requested communication mode of the given user shall be stored
Return code	
`E_OK`	Request is accepted
`E_NOT_OK`	Invalid parameter
`COMM_E_UNINIT`	COMM is not initialized
Functional Description	
This function queries the requested communication mode of the corresponding user.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant.	
Expected Caller Context	
> Function can be called in task and interrupt context	

Table 4-13 ComM_GetRequestedComMode

4.2.11 ComM_GetCurrentComMode

Prototype	
`Std_ReturnType` **`ComM_GetCurrentComMode`** `( ComM_UserHandleType User, ComM_ModeType* ComMode )`	
Parameter	
`User`	Index of the User, the user handles are generated and can be found in the ComM_Cfg.h file
`ComMode`	Pointer where the current communication mode of the given user shall be stored
Return code	
`E_OK`	Request is accepted
`E_NOT_OK`	Invalid parameter
`COMM_E_UNINIT`	COMM is not initialized
Functional Description	
This function queries the current communication mode of the corresponding user. If the user is assigned to more than one communication channel, then always the lowest communication mode is returned.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant.	

Expected Caller Context
> Function can be called in task and interrupt context

Table 4-14 ComM_GetCurrentComMode

4.2.12 ComM_GetCurrentPNCComMode

Prototype	
`Std_ReturnType` **`ComM_GetCurrentPNCComMode`** `( ComM_UserHandleType User, ComM_ModeType* ComMode )`	
Parameter	
`User`	Index of the User, the user handles are generated and can be found in the ComM_Cfg.h file
`ComMode`	Pointer where the current communication mode of the given user shall be stored
Return code	
`E_OK`	Request is accepted
`E_NOT_OK`	Invalid parameter
`COMM_E_UNINIT`	COMM is not initialized
`E_NO_PNC_ASSIGNED`	No PNCs assigned to the user
`E_MULTIPLE_PNC_ASSIGNED`	Multiple PNCs assigned to the user
Functional Description	
This function queries the current communication mode of exactly one PNC assigned to a user.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant.	
Expected Caller Context	
> Function can be called in task and interrupt context	

Table 4-15 ComM_GetCurrentPNCComMode

4.2.13 ComM_PreventWakeUp

Prototype	
`Std_ReturnType` **`ComM_PreventWakeUp`** `( NetworkHandleType Channel, boolean Status )`	
Parameter	
`Channel`	Index of the system channel
`Status`	TRUE: Wake Up Inhibition is switched on FALSE: Wake Up Inhibition is switched off
Return code	
`E_OK`	Request is accepted

`E_NOT_OK`	Request is ignored if one of the following occurs > Channel parameter is invalid or > 'Wake-Up Inhibition Enabled‘ is de-activated in the module configuration or > ‘ECU Group Classification’ does not support Prevent Wake-Up (refer to chapter 4.2.18).
`COMM_E_UNINIT`	COMM is not initialized
Functional Description	
This function changes the inhibition status `ComMNoWakeUp` of the COMM for the given channel.	
Particularities and Limitations	
> A proper ECU Group Classification shall be set before using this API (refer to chapter 4.2.18). > Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant.	
Expected Caller Context	
> Function can be called in task and interrupt context	

Table 4-16 ComM_PreventWakeUp

4.2.14 ComM_LimitChannelToNoComMode

Prototype	
`Std_ReturnType` **`ComM_LimitChannelToNoComMode`** `( NetworkHandleType Channel, boolean Status )`	
Parameter	
`Channel`	Index of the system channel
`Status`	TRUE: limitation to COMM_NO_COMMUNICATION is ON FALSE: limitation to COMM_NO_COMMUNICATION is OFF
Return code	
`E_OK`	Request is accepted
`E_NOT_OK`	Request is ignored if one of the following occurs > Channel parameter is invalid or > 'Mode Limitation Enabled‘ is de-activated in the module configuration or > ‘ECU Group Classification’ does not support Mode Limitation (refer to chapter 4.2.18) or > Nm Variant NONE is configured on the channel.
`COMM_E_UNINIT`	COMM is not initialized
Functional Description	
This function changes the inhibition status `ComMNoCom` of the COMM for the given channel. Update of inhibition status is always performed, independent of the current state but is ignored until the channel is in substate COMM_FULL_COM_NETWORK_REQUESTED.	

Particularities and Limitations
> A proper ECU Group Classification shall be set before using this API (refer to chapter 4.2.18). > Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant.
Expected Caller Context
> Function can be called in task and interrupt context

Table 4-17 ComM_LimitChannelToNoComMode

4.2.15 ComM_LimitECUToNoComMode

Prototype	
`Std_ReturnType` **`ComM_LimitECUToNoComMode`** `( boolean Status )`	
Parameter	
`Status`	TRUE: limitation to COMM_NO_COMMUNICATION is ON FALSE: limitation to COMM_NO_COMMUNICATION is OFF
Return code	
`E_OK`	Request is accepted
`E_NOT_OK`	Request is ignored if one of the following occurs > 'Mode Limitation Enabled‘ is de-activated in the module configuration or > ‘ECU Group Classification’ does not support Mode Limitation (refer to chapter 4.2.18) or > The API ComM_LimitChannelToNoComMode returned E_NOT_OK for at least one channel.
`COMM_E_UNINIT`	COMM is not initialized
Functional Description	
This function changes the inhibition status `ComMNoCom` of the COMM for all channels.	
Particularities and Limitations	
> A proper ECU Group Classification shall be set before using this API (refer to chapter 4.2.18). > Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant.	
Expected Caller Context	
> Function can be called in task and interrupt context	

Table 4-18 ComM_LimitECUToNoComMode

4.2.16 ComM_ReadInhibitCounter

Prototype
`Std_ReturnType` **`ComM_ReadInhibitCounter`** `( uint16* CounterValue )`

Parameter	
`CounterValue`	Pointer where the value of the COMM mode inhibition counter shall be stored
Return code	
`E_OK`	Request is accepted
`E_NOT_OK`	Invalid parameter
`COMM_E_UNINIT`	COMM is not initialized
Functional Description	
This function returns the amount of rejected Full Communication user requests.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant.	
Expected Caller Context	
> Function can be called in task and interrupt context	

Table 4-19 ComM_ReadInhibitCounter

4.2.17 ComM_ResetInhibitCounter

Prototype	
`Std_ReturnType` **`ComM_ReadInhibitCounter`** `( void )`	
Parameter	
–	-
Return code	
`E_OK`	Request is accepted
`COMM_E_UNINIT`	COMM is not initialized
Functional Description	
This function resets the counter of rejected Full Communication user requests.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant.	
Expected Caller Context	
> Function can be called in task and interrupt context	

Table 4-20 ComM_ResetInhibitCounter

4.2.18 ComM_SetECUGroupClassification

Prototype	
`Std_ReturnType` **`ComM_SetECUGroupClassification`** `( ComM_InhibitionStatusType Status )`	
Parameter	
`Status`	Defines Mode Inhibition types the ECU is affected by: 0 - ECU is not affected by mode inhibition 1 - ECU is affected by Wake-up Inhibition only 2 - ECU is affected by Mode Limitation only 3 - ECU is affected by both inhibition types
Return code	
`E_OK`	Request is accepted
`E_NOT_OK`	Invalid parameter
`COMM_E_UNINIT`	COMM is not initialized
Functional Description	
This function changes the ECU group classification status during runtime. The value is stored non-volatile.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant.	
Expected Caller Context	
> Function can be called in task and interrupt context	

Table 4-21 ComM_SetECUGroupClassification

4.2.19 ComM_GetVersionInfo

Prototype	
`void` **`ComM_GetVersionInfo`** `( Std_versionInfoType* versioninfo )`	
Parameter	
`versioninfo`	Pointer where the version information shall be stored.
Return code	
–	-
Functional Description	
This function is called to get the version information of the COMM.	

Particularities and Limitations
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > The Function is only available if it is enabled during pre-compile time (COMM_VERSION_INFO_API == STD_ON)
Expected Caller Context
> Function can be called in task and interrupt context

Table 4-22 ComM_GetVersionInfo

4.2.20 ComM_MainFunction

Prototype	

`void` **`ComM_MainFunction_<Channel_ID>`** `( void )`
`(Channel_ID 0..255)`	

Parameter	
–	-
Return code	
–	-
Functional Description	
This function must be called cyclically with the configured COMM cycle time. Within this function COMM performs the channel specific state transitions and state change notifications to users and BswM.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant.	
Expected Caller Context	
> Function must be called in task context and not in a reentrant way	

Table 4-23 ComM_MainFunction

4.2.21 ComM_GetState

Prototype	
`Std_ReturnType` **`ComM_GetState`** `( NetworkHandleType Channel, ComM_StateType* State )`	
Parameter	
`Channel`	Index of the system channel
`State`	Pointer where the current COMM state shall be stored
Return code	
`E_OK`	Request is accepted

`E_NOT_OK`	Invalid parameter
`COMM_E_UNINIT`	COMM is not initialized
Functional Description	
This function queries the current communication state of the corresponding channel.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant. > COMM must be initialized.	
Expected Caller Context	
> Function can be called in task and interrupt context	

Table 4-24 ComM_GetState

4.2.22 ComM_LimitPncToChannelRouting

Prototype	
`Std_ReturnType` **`ComM_LimitPncToChannelRouting`**`( PNCHandleType Pnc, NetworkHandleType Channel, boolean Status )`	
Parameter	
`Pnc`	Handle of the PNC to set the limitation status for. Handles can be found in ComM_Cfg.h file.
`Channel`	Handle of the system channel to set the limitation status for. Handles can be found in ComM_Cfg.h file.
`Status`	TRUE: activate the Routing Limitation of the PNC on the channel. FALSE: de-activate the Routing Limitation of the PNC on the channel. This is the default status set after initialization of ComM module.
Return code	
`E_OK`	The Routing Limitation status is accepted, parameters are correct and ComM module is initialized.
`E_NOT_OK`	The Routing Limitation status is not accepted if one of following occurs: > ComM module is not initialized or > One of the parameters is out of range or > The ‘Pnc Gateway Type’ of the system channel provided is COMM_GATEWAY_TYPE_NONE.
Functional Description	
The function stores the limitation status for the given PNC and Channel. The status will be used in combination with some other conditions (current Nm state, receiving of ERA signal) to decide whether the routing of PNC information on the channel is active or not. The decision and corresponding actions are taken in the next ComM_MainFunction()	
Particularities and Limitations	
> COMM must be initialized.	

Call context
> Function can be called in task and interrupt context

Table 4-25 ComM_LimitPncToChannelRouting

4.2.23 ComM_GetDcmRequestStatus

Prototype	
`Std_ReturnType` **`ComM_GetDcmRequestStatus`** `(NetworkHandleType Channel, boolean *Status)`	
Parameter	
`Channel [in]`	Valid channel identifier (network handle)
`Status [out]`	Valid pointer where the request status shall be stored TRUE: DCM indicated active diagnostic FALSE: otherwise
Return code	
`E_OK`	Request is accepted
`E_NOT_OK`	Invalid parameter
`COMM_E_UNINIT`	COMM is not initialized
Functional Description	
Queries the status of DCM active diagnostic request of the corresponding channel.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > The function is only available if DCM module is present (COMM_DCM_INDICATION == STD_ON)	
Call context	
> Function can be called in task and interrupt context	

Table 4-26 ComM_GetDcmRequestStatus

4.2.24 ComM_GetMinFullComModeTimerStatus

Prototype	
`Std_ReturnType` **`ComM_GetMinFullComModeTimerStatus`** `(NetworkHandleType Channel, boolean *Status)`	
Parameter	
`Channel [in]`	Valid channel identifier (network handle)
`Status [out]`	Valid pointer where the timer status shall be stored TRUE: Min Full Com Mode Timer is running FALSE: otherwise

Return code	
`E_OK`	Request is accepted
`E_NOT_OK`	Invalid parameter
`COMM_E_UNINIT`	COMM is not initialized
Functional Description	
Queries the status of Min Full Com Mode Timer of the corresponding channel.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant. > The function is only available if at least one channel has Min Full Com Mode Timer (COMM_MINFULLCOMTIMEOFCHANNEL == STD_ON)	
Call context	
> Function can be called in task and interrupt context	

Table 4-27 ComM_GetMinFullComModeTimerStatus

4.2.25 ComM_MainFunction_Satellite

Prototype	

`void` **`ComM_MainFunction_<Id>_<OsAppl>`** `( void )`
`(Channel_ID 0..255)`	

Parameter	
–	-
Return code	
–	-
Functional Description	
This function must be called cyclically with the configured COMM channel specific cycle time. It Implements the synchronization process for master and satellite partitions. It synchronizes the data and performs channel-specific state transitions and state change notifications. This function is only available if Multi-Partition is configured.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant for different partitions. > This function has to be called cyclically on task level by BSW Scheduler > This function has to be called in case of Multi-Partition for each channel. > This function must not be called by the application.	
Expected Caller Context	
> This function can be called from task level only.	

Table 4-28 ComM_MainFunction_Satellite

4.3 Services used by COMM

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

Component	API
DET	`Det_ReportError`
CanSM	`CanSM_RequestComMode`
CanSM	`CanSM_GetCurrentComMode`
LinSM	`LinSM_RequestComMode`
LinSM	`LinSM_GetCurrentComMode`
FrSM	`FrSM_RequestComMode`
FrSM	`FrSM_GetCurrentComMode`
EthSM	`EthSM_RequestComMode`
EthSM	`EthSM_GetCurrentComMode`
NvM	`NvM_GetErrorStatus`
NvM	`NvM_SetRamBlockStatus`
Nm	`Nm_PassiveStartUp`
Nm	`Nm_NetworkRequest`
Nm	`Nm_NetworkRelease`
BswM	`BswM_ComM_CurrentMode`
BswM	`BswM_ComM_CurrentPNCMode`
BswM	`BswM_ComM_InitiateReset`
SchM	`SchM_Enter_ComM_COMM_EXCLUSIVE_AREA_0`
SchM	`SchM_Exit_ComM_COMM_EXCLUSIVE_AREA_0`
SchM	`SchM_Enter_ComM_COMM_EXCLUSIVE_AREA_1`
SchM	`SchM_Exit_ComM_COMM_EXCLUSIVE_AREA_1`
COM	`Com_SendSignal`
COM	`Com_ReceiveSignal`

Table 4-29 Services used by the COMM

4.4 Callback Functions

This chapter describes the callback functions that are implemented by the COMM and can be invoked by other modules. The prototypes of the callback functions are provided in the header files `ComM_BusSM.h`, `ComM_Dcm.h`, `ComM_EcuMBswM.h` and `ComM_Nm.h`.

4.4.1 ComM_CommunicationAllowed

Prototype
`void` **`ComM_CommunicationAllowed`** `( NetworkHandleType Channel, boolean Allowed )`

Parameter	
`Channel`	Index of the system channel
`Allowed`	TRUE: Communication is allowed FALSE: Communication is not allowed (default after COMM initialization)
Return code	
–	-
Functional Description	
The function indicates to COMM when communication is allowed.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is non-reentrant. > COMM must be initialized > The communication allowed state is only evaluated in the COMM state COMM_NO_COM_REQUEST_PENDING	
Expected Caller Context	
> Function can be called in task and interrupt context	

Table 4-30 ComM_CommunicationAllowed

4.4.2 ComM_EcuM_WakeUpIndication

Prototype	
`void` **`ComM_EcuM_WakeUpIndication`** `( const NetworkHandleType Channel )`	
Parameter	
`Channel`	Index of the system channel
Return code	
–	-
Functional Description	
This function notifies the COMM about a valid bus wake-up event. The COMM stores this event and start up the corresponding channel. If the indicated channel is a managed channel, COMM also starts up the referenced managing channel.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is asynchronous. > This function is reentrant. > COMM must be initialized.	
Expected Caller Context	
> Function can be called in task and interrupt context	

Table 4-31 ComM_EcuM_WakeUpIndication

4.4.3 ComM_EcuM_PNCWakeUpIndication

Prototype	
`void` **`ComM_EcuM_PNCWakeUpIndication`** `( const PNCHandleType PNCid )`	
Parameter	
`PNCid`	ID of the Partial Network Cluster
Return code	
–	-
Functional Description	
This function notifies the COMM about a valid bus wake-up event. The COMM stores this event and start up the corresponding pnc.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is asynchronous. > This function is reentrant. > COMM must be initialized.	
Expected Caller Context	
> Function can be called in task and interrupt context	

Table 4-32 ComM_EcuM_PNCWakeUpIndication

4.4.4 ComM_BusSM_ModeIndication

Prototype	
`void` **`ComM_BusSM_ModeIndication`** `( const NetworkHandleType Channel, ComM_ModeType* ComM_Mode )`	
Parameter	
`Channel`	Index of the system channel
`ComM_Mode`	Pointer to variable which contains the new BusSM communication mode
Return code	
–	-
Functional Description	
This function notifies the COMM about a state change of the BusSM. The COMM performs corresponding actions dependent on the given ComM_Mode.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is asynchronous. > This function is reentrant. > COMM must be initialized.	
Expected Caller Context	
> Function can be called in task and interrupt context	

Table 4-33 ComM_BusSM_ModeIndication

4.4.5 ComM_BusSM_BusSleepMode

Prototype	
void **ComM_BusSM_BusSleepMode** (NetworkHandleType Channel)	
Parameter	
Channel	Index of the system channel
Return code	
–	-
Functional Description	
This function notifies COMM from the corresponding Bus State Manager that the actual mode is Bus Sleep. Only applicable for LINSLAVE nodes. The COMM uses this indication for transitioning to COMM_NO_COMMUNICATION.	
Particularities and Limitations	
> The function is available if at least one channel has variant LINSLAVE. > The function is synchronous (Notification). The corresponding state transition from mode Full Communication to No Communication is performed in the next main function cycle. > This function is reentrant. > COMM must be initialized.	
Expected Caller Context	
> Function can be called in task and interrupt context	

Table 4-34 ComM_BusSM_BusSleepMode

4.4.6 ComM_DCM_ActiveDiagnostic

Prototype	
void **ComM_DCM_ActiveDiagnostic** (NetworkHandleType Channel)	
Parameter	
Channel	Index of the system channel
Return code	
–	-
Functional Description	
This function notifies the COMM about the start of an active diagnostic session for the given channel. The COMM starts the communication for this channel as long as the DCM informs the COMM about the end of this session. If more channels needed for diagnostic purpose, DCM needs to indicate it for each channel. If the Nm Variant configured on the channel is PASSIVE or LINSLAVE > COMM ignores the indication and > Reports a DET error with error code COMM_E_DIAGNOSTIC_NOT_SUPPORTED.	

Particularities and Limitations
> Service ID: see table 'Service IDs' > This function is asynchronous. > This function is reentrant. > COMM must be initialized.
Expected Caller Context
> Function can be called in task and interrupt context

Table 4-35 ComM_DCM_ActiveDiagnostic

4.4.7 ComM_DCM_InactiveDiagnostic

Prototype	
`void` **`ComM_DCM_InactiveDiagnostic`** `(NetworkHandleType Channel )`	
Parameter	
`Channel`	Index of the system channel
Return code	
–	-
Functional Description	
This function notifies the COMM about the end of the DCM diagnostic session for the given channel. The COMM triggers the network shutdown for this channel if all COMM users assigned to it request the COMM state No Communication. If the Nm Variant configured on the channel is PASSIVE or LINSLAVE > COMM ignores the indication and > Reports a DET error with error code COMM_E_DIAGNOSTIC_NOT_SUPPORTED.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is asynchronous. > This function is reentrant. > COMM must be initialized.	
Expected Caller Context	
> Function can be called in task and interrupt context	

Table 4-36 ComM_DCM_InactiveDiagnostic

4.4.8 ComM_Nm_NetworkStartIndication

Prototype	
`void` **`ComM_Nm_NetworkStartIndication`** `( NetworkHandleType Channel )`	
Parameter	
`Channel`	Index of the system channel, which has already entered Bus-Sleep Mode

Return code	
–	-
Functional Description	
This function notifies the COMM about a restart of the network management. The restart was triggered by receiving an Nm message when Nm was in Bus-Sleep Mode. COMM stores the event and starts up the corresponding network in the next ComM_MainFunction.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is asynchronous. > This function is reentrant. > COMM must be initialized.	
Expected Caller Context	
> Function can be called in task and interrupt context	

Table 4-37 ComM_Nm_NetworkStartIndication

4.4.9 ComM_Nm_NetworkMode

Prototype	
`void` **`ComM_Nm_NetworkMode`** `( NetworkHandleType Channel )`	
Parameter	
`Channel`	Index of the system channel
Return code	
–	-
Functional Description	
This function notifies the COMM that the Nm entered the Network Mode.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is asynchronous. > This function is reentrant. > COMM must be initialized.	
Expected Caller Context	
> Function can be called in task and interrupt context	

Table 4-38 ComM_Nm_NetworkMode

4.4.10 ComM_Nm_PrepareBusSleep

Prototype	
`void` **`ComM_Nm_PrepareBusSleep`** `( NetworkHandleType Channel )`	
Parameter	
`Channel`	Index of the system channel

Return code	
–	-
Functional Description	
This function notifies the COMM that the NM has entered Prepare Bus-Sleep Mode. The COMM uses this function as synchronization for the network shutdown. Inside this function the COMM sets the corresponding Bus state Manager into the COMM mode Silent Communication and the COMM itself changes the state to Silent Communication.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant (but not for the same Nm channel). > COMM must be initialized.	
Expected Caller Context	
> Function can be called in task and interrupt context	

Table 4-39 ComM_Nm_PrepareBusSleep

4.4.11 ComM_Nm_BusSleepMode

Prototype	
`void ComM_Nm_BusSleepMode ( NetworkHandleType Channel )`	
Parameter	
`Channel`	Index of the system channel
Return code	
–	-
Functional Description	
This function notifies the COMM that the NM ends the prepare bus sleep phase and has entered Bus-Sleep Mode. The COMM uses this function as synchronization for the network shutdown. Inside this function the COMM sets the corresponding Bus state Manager into the COMM mode No Communication and the COMM itself changes the state to No Communication.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is synchronous. > This function is reentrant (but not for the same Nm channel). > COMM must be initialized.	
Expected Caller Context	
> Function can be called in task and interrupt context	

Table 4-40 ComM_Nm_BusSleepMode

4.4.12 ComM_Nm_RestartIndication

Prototype	
`void ComM_Nm_RestartIndication ( NetworkHandleType Channel )`	
Parameter	
`Channel`	Index of the system channel, which has already entered Bus-Sleep Mode
Return code	
–	-
Functional Description	
NmIf notifies COMM that NmIf has started to shut down the coordinated busses, and not all coordinated busses have indicated Bus-Sleep Mode, and on at least one of the coordinated busses Nm is restarted. COMM stores the event and starts up the corresponding network in the next ComM_MainFunction.	
Particularities and Limitations	
> Service ID: see table 'Service IDs' > This function is asynchronous. > This function is reentrant. > COMM must be initialized.	
Expected Caller Context	
> Function can be called in task and interrupt context	

Table 4-41 ComM_Nm_RestartIndication

4.4.13 ComM_Nm_StateChangeNotification

Prototype	
`void ComM_Nm_StateChangeNotification ( const NetworkHandleType Channel, const Nm_StateType NmPreviousState, const Nm_StateType NmCurrentState )`	
Parameter	
`Channel`	Valid channel identifier (network handle)
`NmPreviousState`	Previous state of Nm
`NmCurrentState`	Current state of Nm
Return code	
–	-
Functional Description	
Notification that the Nm state has changed. The Pnc Routing Limitation state is updated depending on Nm has left or entered the state NM_STATE_REPEAT_MESSAGE.	
Particularities and Limitations	
> The function is available if Pnc to Channel Routing Limitation feature is activated. > This function is synchronous. > This function is not reentrant.	
Call context	
> Function can be called in task or interrupt context	

Table 4-42 ComM_Nm_StateChangeNotification

4.4.14 ComM_Nm_ForwardSynchronizedPncShutdown

Prototype	
`void` **`ComM_Nm_ForwardSynchronizedPncShutdown`** `( const NetworkHandleType Channel, const uint8* PncBitVectorPtr )`	
Parameter	
`Channel`	Valid channel identifier
`PncBitVectorPtr`	Pointer to a PNC Bit vector
Return code	
–	-
Functional Description	
NM indicates to COMM that a synchronized PNC shutdown was requested by a Top-Level PNC coordinator (received on a passively coordinated channel), which needs to be forwarded to all actively coordinated channels of the indicated PNCs. The PNCs to be shutdown are indicated via the PncBitVectorPtr (bit set to 1 for each affected PNC). COMM stores the request to forward the synchronized PNC shutdown for the affected PNCs and will forward the synchronized PNC shutdown on all corresponding actively coordinated channels as soon as the PNC is released and enters COMM_PNC_READY_SLEEP state.	
Particularities and Limitations	
> The function is only available if 'Synchronized Pnc Shutdown Enabled' is activated. > This function is synchronous. > This function is reentrant. > COMM must be initialized.	
Call Context	
> Function can be called in task and interrupt context	

Table 4-43 ComM_Nm_ForwardSynchronizedPncShutdown

4.4.15 ComM_ComCbk_<SignalName>

Prototype	
`void` **`ComM_ComCbk_<SignalName>`** `( void )`	
Parameter	
–	-
Return code	
–	-
Functional Description	
Notification that ComSignal data which is used to transport the partial network channel request information has changed. SignalName is the name of the corresponding EIRA_RX or ERA ComSignal. The function is generated.	
Particularities and Limitations	
> The function is available if support of Partial Networking is enabled.	

> This function is synchronous. > This function is not reentrant.
Call context
> Function can be called in task context

Table 4-44 ComM_ComCbk_<SignalName>

4.5 Configurable Interfaces

4.5.1 Notifications

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

4.5.1.1 Dcm_ComM_FullComModeEntered

Prototype	
`void` **`Dcm_ComM_FullComModeEntered`** `( NetworkHandleType Channel )`	
Parameter	
`Channel`	Index of the system channel
Return code	
–	-
Functional Description	
This callback function informs the DCM about the COMM state change into Full Communication.	
Particularities and Limitations	
> This callback function is only available if the DCM module is activated in the ECU configuration.	
Call context	
> The function is called in the context of ComM_BusSM_ModeIndication	

Table 4-45 Dcm_ComM_FullComModeEntered

4.5.1.2 Dcm_ComM_SilentComModeEntered

Prototype	
`void` **`Dcm_ComM_SilentComModeEntered`** `( NetworkHandleType Channel )`	
Parameter	
`Channel`	Index of the system channel
Return code	
–	-
Functional Description	
This callback function informs the DCM about the COMM state change into Silent Communication.	
Particularities and Limitations	
> This callback function is only available if the DCM module is activated in the ECU configuration.	
Call context	
> The function is called in the context of ComM_BusSM_ModeIndication	

Table 4-46 Dcm_ComM_SilentComModeEntered

4.5.1.3 Dcm_ComM_NoComModeEntered

Prototype	
`void` **`Dcm_ComM_NoComModeEntered`** `( NetworkHandleType Channel )`	
Parameter	
`Channel`	Index of the system channel
Return code	
–	-
Functional Description	
This callback function informs the DCM about the COMM state change into No Communication.	
Particularities and Limitations	
> This callback function is only available if the DCM module is activated in the ECU configuration.	
Call context	
> The function is called in the context of ComM_BusSM_ModeIndication	

Table 4-47 Dcm_ComM_NoComModeEntered

4.5.1.4 BswM_ComM_CurrentMode

Prototype	
`void` **`BswM_ComM_CurrentMode`** `( NetworkHandleType Network, ComM_ModeType RequestedMode )`	
Parameter	
`Network`	Index of the system channel
`RequestedMode`	Current Communication Mode, where COMM changed to
Return code	
–	-
Functional Description	
COMM indicates every main state change to BswM.	
Particularities and Limitations	
> -	
Call context	
> The function is called in the context of ComM_BusSM_ModeIndication	

Table 4-48 BswM_ComM_CurrentMode

4.5.1.5 BswM_ComM_CurrentPNCMode

Prototype
`void` **`BswM_ComM_CurrentPNCMode`** `( PNCHandleType Pnc, ComM_PncModeType RequestedMode )`

Parameter	
`Pnc`	Partial network identifier
`RequestedMode`	Partial network state where the COMM changed to
Return code	
–	-
Functional Description	
COMM indicates every partial network state change to BswM.	
Particularities and Limitations	
> This callback function is only available if Partial Network functionality is activated in the ECU configuration.	
Call context	
> The function is called in the context of ComM_MainFunction	

Table 4-49 BswM_ComM_CurrentPNCMode

4.5.1.6 BswM_ComM_InitiateReset

Prototype	
`void` **`BswM_ComM_InitiateReset`** `( void )`	
Parameter	
–	-
Return code	
–	-
Functional Description	
COMM indicates a need for an ECU reset to BswM, see chapter 'Mode Limitation to NO_COM' for details.	
Particularities and Limitations	
> This callback function is only available if BswM has a Mode Request Port with the Source BswMComMInitiateReset.	
Call context	
> The function is called in the context of ComM_BusSM_ModeIndication	

Table 4-50 BswM_ComM_InitiateReset

4.5.1.7 Rte_Switch_ComM_<UserName>_currentMode

Prototype
`Std_ReturnType` **`Rte_Switch_ComM_<UserName>_currentMode`** `(Rte_ModeType_ComMMode mode)`

Parameter	
`mode`	> RTE_MODE_ComMMode_NO_COMMUNICATION, no communication is entered > RTE_MODE_ComMMode_SILENT_COMMUNICATION, silent communication is entered > RTE_MODE_ComMMode_FULL_COMMUNICATION, full communication is entered
Return code	
`Std_ReturnType`	> RTE_E_OK, the SW-C notified > RTE_E_LIMIT, the SW-C does not notified the mode and the COMM shall informed again
Functional Description	
This callback functions inform the SW-C about a mode change of a COMM user.	
Particularities and Limitations	
> This callback function is only available for users with parameter ComMUserModeNotification set to true.	
Call context	
> The function is called in the context of ComM_MainFunction	

Table 4-51 Rte_Switch_ComM_<UserName>_currentMode

5 Service Ports

5.1.1 Client Server Interface

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

5.1.1.1 Provide Ports on COMM Side

At the Provide Ports of the COMM the API functions described in chapter 4.2 are available as Runnable Entities. The Runnable Entities are invoked via Operations. The mapping from a SWC client call to an Operation is performed by the RTE. In this mapping the RTE adds Port Defined Argument Values to the client call of the SWC, if configured.

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

5.1.1.1.1 ComM_UserRequest

Operation	API Function	Port Defined Argument Values
RequestComMode	ComM_RequestComMode	ComM_UserHandleType UserHandle
GetCurrentComMode	ComM_GetCurrentComMode	ComM_UserHandleType UserHandle
GetMaxComMode	ComM_GetMaxComMode	ComM_UserHandleType UserHandle
GetRequestedComMode	ComM_GetRequestedComMode	ComM_UserHandleType UserHandle

Table 5-1 ComM_UserRequest

The naming rule for corresponding ports is UR_<user_name>, e.g. UR_ComMUser_000.

5.1.1.1.2 ComM_ECUModeLimitation

Operation	API Function
LimitECUToNoComMode	ComM_LimitECUToNoComMode
ReadInhibitCounter	ComM_ReadInhibitCounter
ResetInhibitCounter	ComM_ResetInhibitCounter
SetECUGroupClassification	ComM_SetECUGroupClassification

Table 5-2 ComM_ECUModeLimitation

The naming rule for the corresponding port is modeLimitation.

5.1.1.1.3 ComM_ChannelWakeUp

Operation	API Function	Port Defined Argument Values
PreventWakeUp	ComM_PreventWakeUp	NetworkHandleType Channel
GetInhibitionStatus (optional, see below)	ComM_GetInhibitionStatus	NetworkHandleType Channel

Table 5-3 ComM_ChannelWakeUp

For compatibility, the operation GetInhibitionStatus can be omitted from this interface. Its presence depends on the value of the optional configuration parameter ComMOperationGetInhibitionStatusEnabled:

> If the parameter does not exist or is set to ‘true’, the interface contains the operation GetInhibitionStatus.
> If the parameter exists and is set to ‘false’, the interface does not expose the operation GetInhibitionStatus.

Note that the COMM API Function ComM_GetInhibitionStatus exists in both cases and is also exposed through the Client Server Interface ComM_ChannelLimitation as described in chapter 5.1.1.1.4.

The naming rule for corresponding ports is CW_<channel_name>, e.g. CW_ComMChannel_CAN0.

5.1.1.1.4 ComM_ChannelLimitation

Operation	API Function	Port Defined Argument Values
LimitChannelToNoComMode	ComM_LimitChannelToNoComMode	NetworkHandleType Channel
GetInhibitionStatus	ComM_GetInhibitionStatus	NetworkHandleType Channel

Table 5-4 ComM_ChannelLimitation

The naming rule for corresponding ports is CL_<channel_name>, e.g. CL_ComMChannel_CAN0.

5.1.1.2 Require Ports on COMM Side

COMM does not require any Ports providing Client Server Interface.

5.1.2 Mode Switch Interface

5.1.2.1 ComM_CurrentMode

The interface is optional. It can be activated or de-activated for each configured COMM user separately using the parameter ComMUserModeNotification.

The purpose of this interface is to inform an SW-C about the current COMM mode for each configured COMM user, to which an SW-C is connected. For each configured interface COMM requires a notification callback function, which is provided by the RTE and described in 4.5.1.7.

Operation	Rte Interface	Mode Declaration Group
currentMode	Rte_Switch_ComM_UM_<UserName>_currentMode e.g. Rte_Switch_ComM_UM_ComMUser_000_currentMode	RTE_MODE_ComMMode_COMM_FULL_COMMUNICATION RTE_MODE_ComMMode_COMM_NO_COMMUNICATION RTE_MODE_ComMMode_COMM_SILENT_COMMUNICATION

Table 5-5 ComM_CurrentMode

The naming rule for corresponding ports is UM_<user_name>, e.g. UM_ComMUser_000.

5.1.3 Sender Receiver Interface

5.1.3.1 ComM_CurrentChannelRequest

The interface is optional. It can be activated or de-activated for each configured COMM channel separately using the parameter ComMFullCommRequestNotificationEnabled.

The purpose of this interface is to inform an SW-C about COMM users requesting Full Communication for a channel. Whenever the set of COMM users that are currently requesting Full Communication for a channel changes, COMM updates the data element ComM_FullComRequesters_CR_<channel_name>. A change updates the data element only, when COMM accepts the communication request of the COMM user. If a Mode Inhibition is active on a channel, this set is empty because no user is allowed to keep the communication on the channel awake.

Rte Interface	Data element
Rte_Write_ComM_CR_<channel_name>_fullComRequestors	ComM_UserHandleArrayType_<channel_name>

Table 5-6 ComM_CurrentChannelRequest

The type ComM_UserHandleArrayType_<channel_name> exists for each channel where the sender receiver interface is enabled. Refer to the Table 4-3 for details.

Please note that COMM only informs about COMM users requesting Full Communication for users which are directly assigned to the COMM channel. COMM will not inform about COMM users requesting a Partial Network, even if the channel is in Full Communication mode because the Partial Network is requested by such a COMM user.

The naming rule for corresponding ports is CR_<channel_name>, e.g. CR_ComMChannel_CAN0.

Example

Assumptions:

One CAN channel with enabled interface ComM_CurrentChannelRequest, Channel ID '0' and Channel name 'CAN0'.

There are 2 COMM users configured on the channel 'CAN0' having user handles:

> ComMConf_ComMUser_000 (value = 0) and

> ComMConf_ComMUser_001 (value = 1).

The corresponding define macro is `COMM_MAX_CR_CAN0 = 2`

Example Sequence:

The channel is in COMM_NO_COMMUNICATION mode. The application calls

```
ComM_RequestComMode(ComMConf_ComMUser_000,
   COMM_FULL_COMMUNICATION)
```

COMM will call the following RTE interface in the next ComM_MainFunction_0():

```
Rte_Write_ComM_CR_CAN0_fullComRequestors(
   ComM_FullComRequesters_CR_CAN0)
```

The structure passed to the interface contains the following values:

```
/* a single user keeps the channel requested */
ComM_FullComRequesters_CR_CAN0.numberOfRequesters = 1
/* user with handle ComMConf_ComMUser_000 keeps the channel
   requested */
ComM_FullComRequesters_CR_CAN0.handleArray[0] =
   ComMConf_ComMUser_000
/* the 2nd element contains the invalid user handle */
ComM_FullComRequesters_CR_CAN0.handleArray[1] = 0xff
```

6 Abbreviations

6.1 Abbreviations

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

Table 6-1 Abbreviations

7 Contact

Visit our website for more information on

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

www.vector.com