# MICROSAR Communication Manager

Technical Reference

Version 3.14

| Authors | Thomas Petrus |
|---------|---------------|
| Status | Released |

# 1   Document Information

## 1.1   History

| Author | Date | Version | Remarks |
|---|---|---|---|
| Thomas Petrus | 2008-02-01 | 3.0 | Complete rework and adaptation to new template |
| Thomas Petrus | 2008-02-20 | 3.1 | ■ correct ComM_MainFunction() description<br>■ correct and advance ComM service port description<br>■ advance configuration setting description |
| Thomas Petrus | 2008-03-20 | 3.2 | ■ Update Compiler Abstraction and Memory Mapping |
| Thomas Petrus | 2008-03-30 | 3.3 | ■ add description for DEM support<br>■ update file structure<br>■ remove callback decriptions for ComM_LinSm_ModeIndication, ComM_CanSm_ModeIndication and ComM_FrSm_ModeIndication<br>■ add callback description for ComM_BusSM_ModeIndication<br>■ update configuration chapter |
| Thomas Petrus | 2008-05-08 | 3.4 | ■ update include structure<br>■ update chapter 5.1.2 Dynamic Files<br>■ Add chapter 4.4.2, 4.4.3, 5.4 and 5.5<br>■ update configuration description |
| Thomas Kuhl | 2008-08-22 | 3.5 | ■ Update function description of ComM_GetCurrentComMode<br>■ Update function description of ComM_BusSM_ModeInidication<br>■ Update Chapter 8.2. Limitations<br>■ Update Configuration chapter<br>■ Update ComM state diagram |

| | | | | |
|---|---|---|---|---|
| | | | | (ESCAN00029568) |
| Thomas Kuhl | 2008-10-17 | 3.6 | | ■ Update configuration chapter<br>■ Add API description for ComM_Dcm_SetPassiveMode |
| Thomas Kuhl | 2008-12-04 | 3.7 | | ■ Remove ComM_RTE_ComMModeIndication callback function<br>■ Update component history<br>■ Update configuration chapter<br>■ Update chapter 8<br>■ Update chapter 4.1<br>■ Add function description for mode indication (6.6.4, 6.6.5)<br>■ Add chapter 6.7.1.2 |
| Thomas Kuhl | 2009-03-16 | 3.8 | | ■ Remove API ComM_Dcm_SetEcuPassiveMode<br>■ Add Nm Variant OSEK |
| Thomas Kuhl | 2009-07-14 | 3.9 | | ■ Update chapter 5.4 Critical code sections<br>■ Update API descriptions inside chapter 6.3, especially adaption of API return values<br>■ Add chapter 8.1.4 ComM Service API Return Value COMM_UNINIT |
| Thomas Kuhl | 2009-11-19 | 3.10 | | ■ Update chapter 7.1.2 General Configuration<br>■ Update chapter 5.3 Compiler Abstraction and Memory Mapping<br>■ Update chapter 5.5 Handling of non-volatile data<br>■ Update chapter 6.2 Type Definitions |
| Thomas Kuhl | 2010-02-15 | 3.11 | | ■ Add chapter 6.3.17 ComM_ConmmunicationControl<br>■ Add chapter 6.3.18 ComM_GetNumSendRcvAppl<br>■ Add chapter 6.3.19 ComM_GetOverallBusState<br>■ Advance chapter 6.7.1.1 |
| Thomas Kuhl | 2010-04-28 | 3.12 | | ■ Remove chapter 6.3.17 – 6.3.19<br>■ Remove port description of ComM_GetNumSendRcvAppl |

| | | | ■ ESCAN00041031,<br>ESCAN00040931 and<br>ESCAN00039887 |
| --- | --- | --- | --- |
| Thomas Kuhl | 2010-08-11 | 3.13 | ■ ESCAN00043830<br>■ Add description for Ethernet support |
| Thomas Kuhl | 2011-02-10 | 3.14 | ■ Extend description of ComM_Init and ComM_MainFunction<br>■ Add description of BswM_ComM_CurrentMode<br>■ ESCAN00045289, ESCAN00046463 |

Table 1-1   History of the document

## 1.2   Reference Documents

| No. | Title | Version |
| --- | --- | --- |
| [1] | AUTOSAR_ComM.pdf | V2.0.0 |
| [2] | AUTOSAR_SWS_DET.pdf | V2.2.0 |
| [3] | AUTOSAR_SWS_DEM.pdf | V2.2.0 |
| [4] | AUTOSAR_BasicSoftwareModules.pdf | V1.2.0 |
| [5] | AUTOSAR_SWS_DCM.pdf | V3.0.0 |
| [6] | AUTOSAR_SWS_CAN_StateManager.pdf | V1.0.0 |
| [7] | AUTOSAR_SWS_ECU_Statemanager.pdf | V1.2.0 |
| [8] | AUTOSAR_SWS_FlexRay_StateManager.pdf | V1.0.0 |
| [9] | AUTOSAR_SWS_LIN_StateManager.pdf | V1.0.0 |
| [10] | AUTOSAR_SWS_NMInterface.pdf | V1.0.0 |
| [11] | AUTOSAR_SWS_NVRAMManager.pdf | V2.2.0 |
| [12] | AN-ISC-8-1118 MICROSAR BSW Compatibility Check | V1.0.0 |

Table 1-2   Reference documents

## 1.1   Scope of the Document

This technical reference describes the general use of the Communication Manager basic software.

| | |
|---|---|
| ⚠️ | **Please note**<br>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

based on template version 3.1

## Illustrations

## Tables

# 2 Component History

| Component Version | New Features |
|---|---|
| 4.00.00 | Rework for AUTOSAR Release 3 |
| 4.10.00 | ComM_RTE_ComMModeIndication function replaced by ComM user mode indication functions |

Table 2-1   Component history

# 3   Introduction

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

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

\* For the precise AUTOSAR Release 3.x please see the release specific documentation.

The Communication Manager is a resource manager and encapsulates the control of the underlying communication services. He is responsible to:

- coordinate different wake up events independent of the used bus system (e.g. CAN or LIN)

- simplify the resource management by allocating all resources which are necessary  to start/stop communication

- simplify the handling of the underlying communication stack (e.g. network management handling)

- provision of a common behavior and handling of the underlying communication stack.

## 3.1 Architecture Overview

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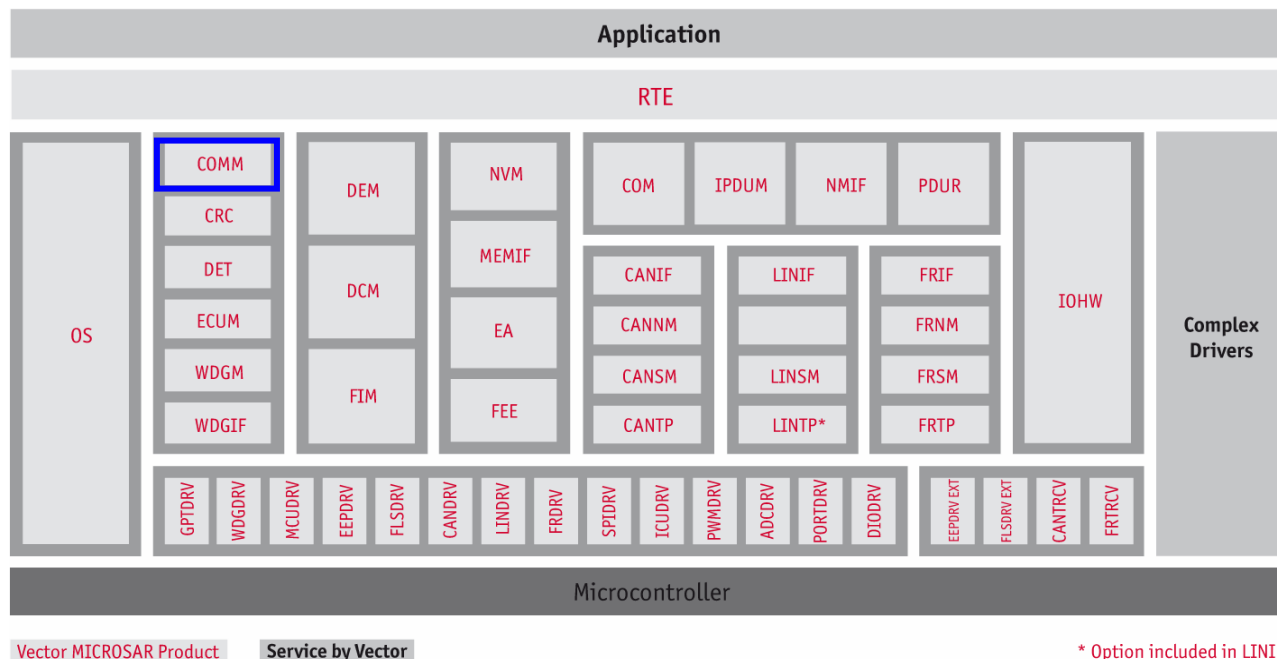


Figure 3-1 AUTOSAR architecture

The next figure shows the interfaces to adjacent modules of the ComM. These interfaces are described in chapter 6.
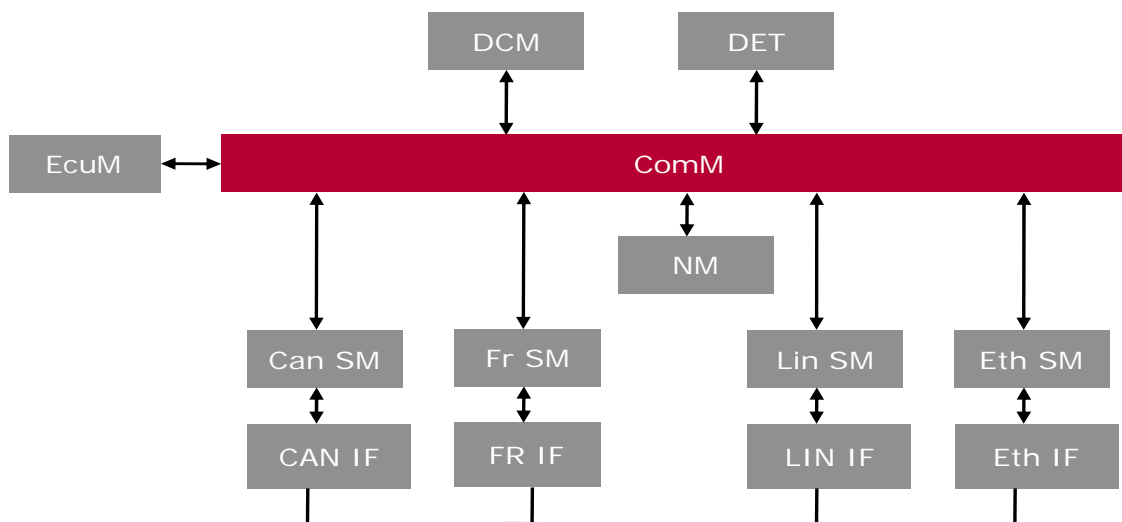


Figure 3-2 Interfaces to adjacent modules of the ComM

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 6.7 and are defined in [1].

# 4 Functional Description

## 4.1 Features

The features listed in this chapter cover the complete functionality specified in [1].

The "supported" and "not supported" features are presented in the following two tables. For further information of not supported features also see chapter 8.

The following features described in [1] are supported:

| Feature |
| --- |
| Communication Control Handling |
| Service Port: ComM_RequestMode |
| NM Variant Handling: FULL |
| NM Variant Handling: LIGHT |
| NM Variant Handling: NONE |
| NM Variant Handling: PASSIVE |
| Communication Inhibition |
| Bus Wake Up Inhibition |
| Service Port: ComM_ChannelWakeUp |
| Service Port: ComM_ChannelLimitation |
| Service Port: ComM_ECUModeLimitation |
| Service Port: ComM_CurrentMode |
| DEM error reporting |
| Synchronous wake up of channels |
| Storage of non-volatile values |

Table 4-1   Supported SWS features

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

| Feature |
| --- |
| ComM bus type: INTERNAL |

Table 4-2   Not supported SWS features

## 4.2 Initialization

For a correct use the Communication Manager must be initialized. At first the ComM_InitMemory() must be called which sets the global state to uninitialized. Afterwards the API function ComM_Init must be called which initializes the Communication Manager.

## 4.3 States
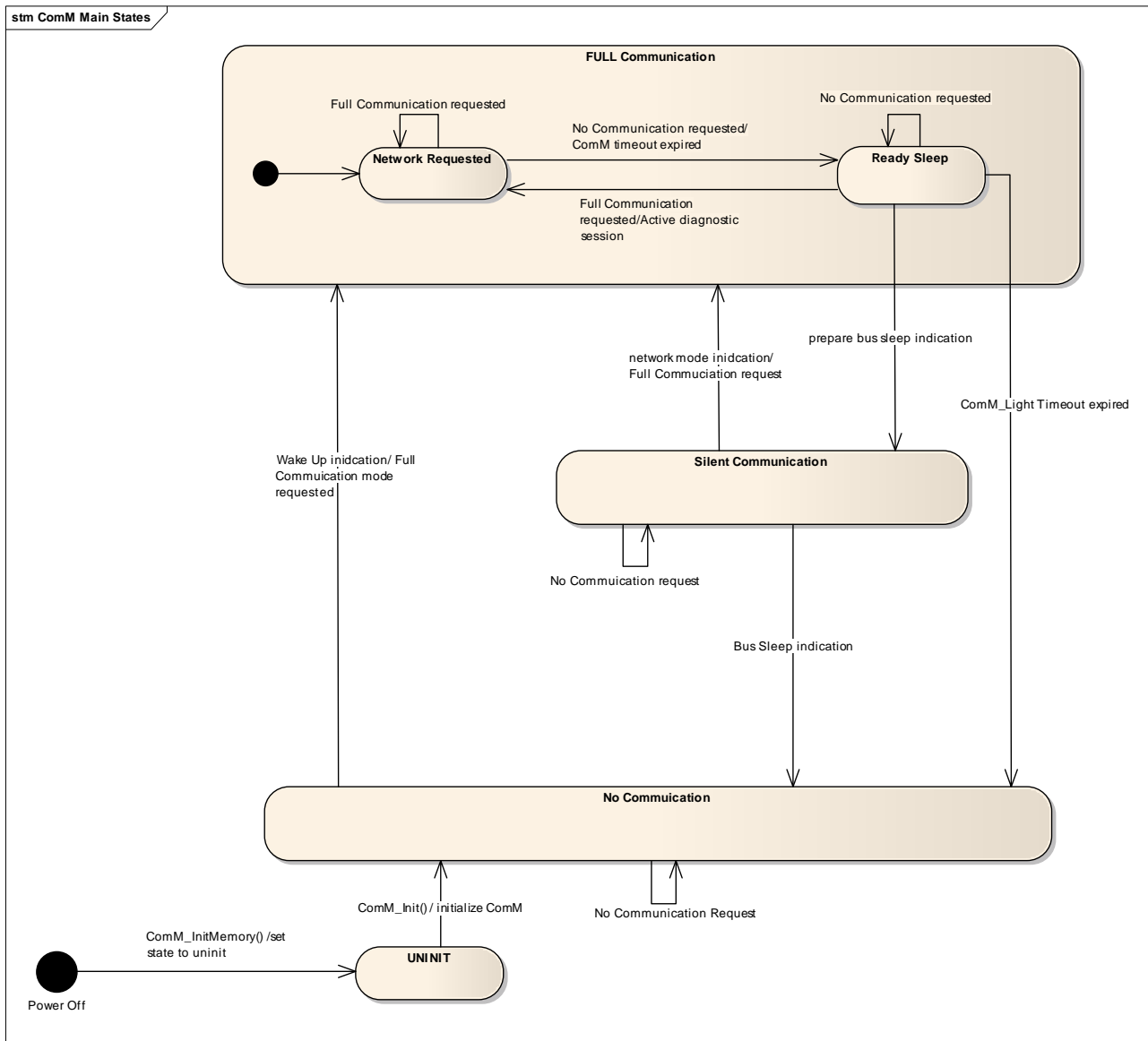


Figure 4-1 state machine of the ComM

Like shown in Figure 4-1 the ComM has a state machine with three global states and the state machine exists per network. The states are representing the abstract status of communication per network.

**UNINIT:**

Before the ComM is initialized it stays in this state. The ComM functionality can not be used.

**No Communication:**

This state is the default state after the initialization of the ComM. The capability of communication is off.

The ComM resides in this state until a user requests the state Full Communication or the EcuM informs the ComM about an external wake up event.

**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 starts the sleep process and changes into the prepare bus sleep mode.

It is not possible to request this state directly via a user.

The ComM resides in this state until:

- the network management signalizes a restart via receiving a NM message
- or a user requests Full Communication again.

Inside this state it is only possible to receive messages. The transmit capability is switched off via the responsible bus state manager.

**Full Communication:**

The state Full Communication represents the highest state of the ComM. Inside this state the full communication capability is available. The state consists of two sub-states:

**Network Requested:**

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

- NM Variant "FULL":
  - The network management is set into the "Normal Operation" state.
  - The ComM resides in this state until all user request No Communication.
- NM Variant "OSEK":
  - The network management is set into the "Normal" state.
  - The ComM starts the "Minimum Full Communication Mode Time" to ensure a minimum active time of the network
  - The ComM resides in this state until all user request No communication and the "Minimum Full Communication Mode Time" is expired.

- NM Variant "Light" and "NONE":
  - The ComM starts the "Minimum Full Communication Mode Time".

- the ComM resides in this state until all user request No Communication and the "Minimum Full Communication Mode Time" is expired.

**Ready Sleep:**

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

- NM Variant "FULL" and "OSEK":

  - The network management is set into the "Ready Sleep" state.

  - The ComM resides in this state until the NM cancels the sleep process or a user requests Full Communication again.

- NM Variant "LIGHT"

  - The ComM starts the "Ready Sleep" timer.

  - He resides in this state until the "Ready Sleep" timer expires or a user requests again Full Communication.

- NM Variant "NONE"

  - This state is not available for this NM variant.

## 4.4 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.

### 4.4.1 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 and user 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

- PASSIVE, AUTOSAR Nm is used but the ECU is not allowed to keep the network awake

- OSEK, NmOsek(direct) is used

- LIGHT, no AUTOSAR Nm is used but the shutdown is synchronized via timeouts

- and NONE, no AUTOSAR Nm is used and no shutdown synchronization are available.

The following sequence diagrams show startup and shutdown of the networks in dependency of the wake up event and the configured Nm variant.
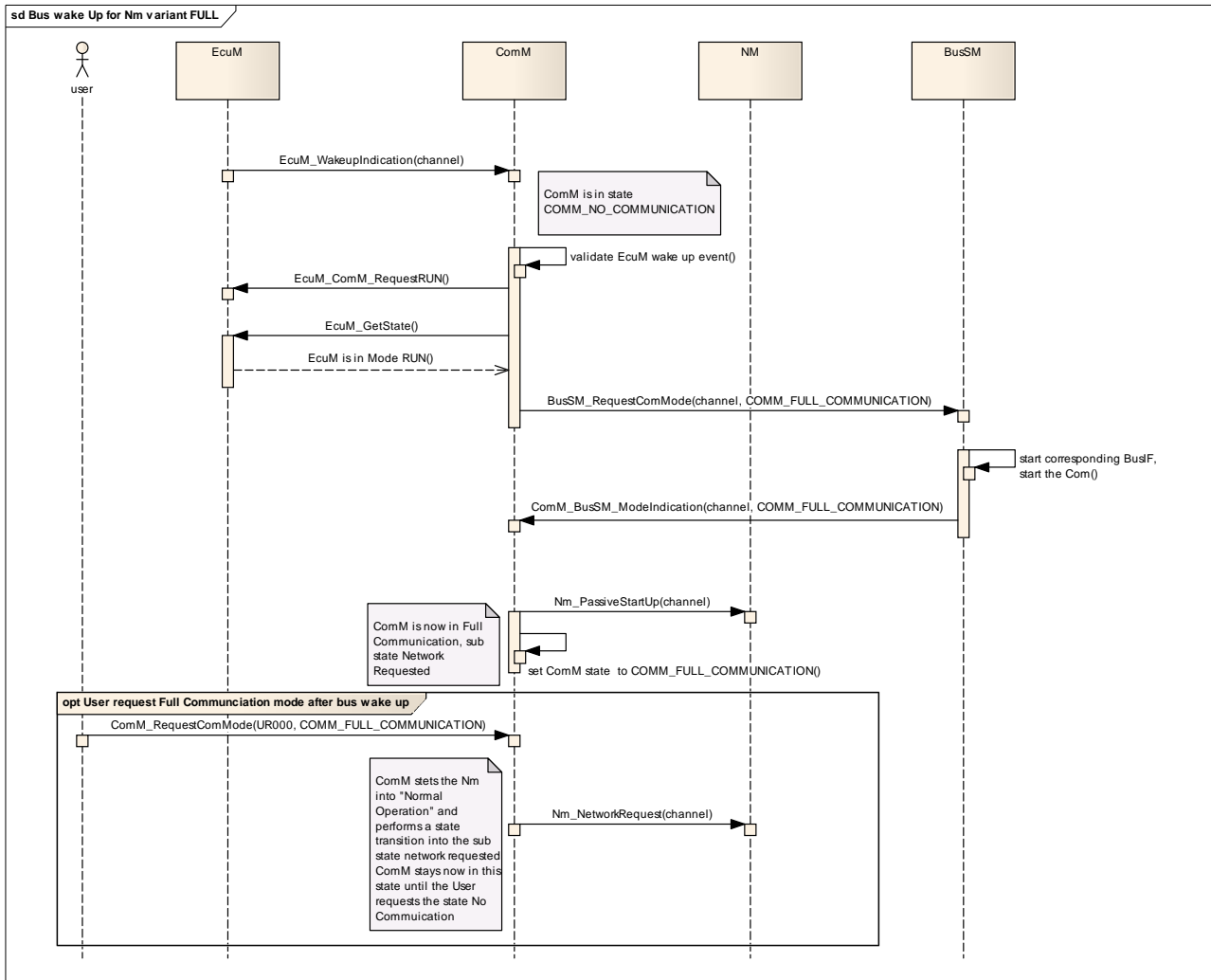
Figure 4-2 Bus wake up with Nm Variant FULL
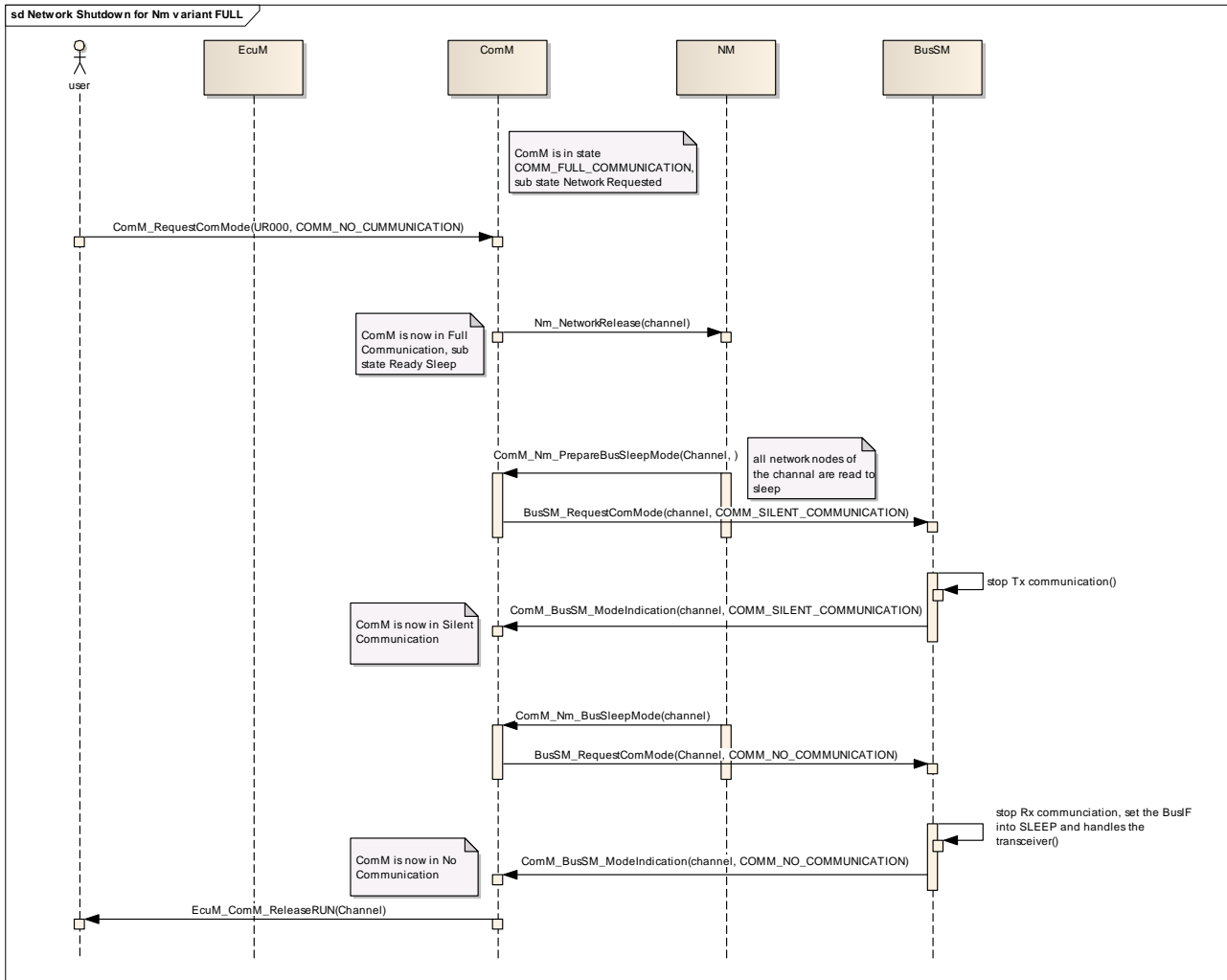
Figure 4-3 Network Shutdown for Nm variant Full
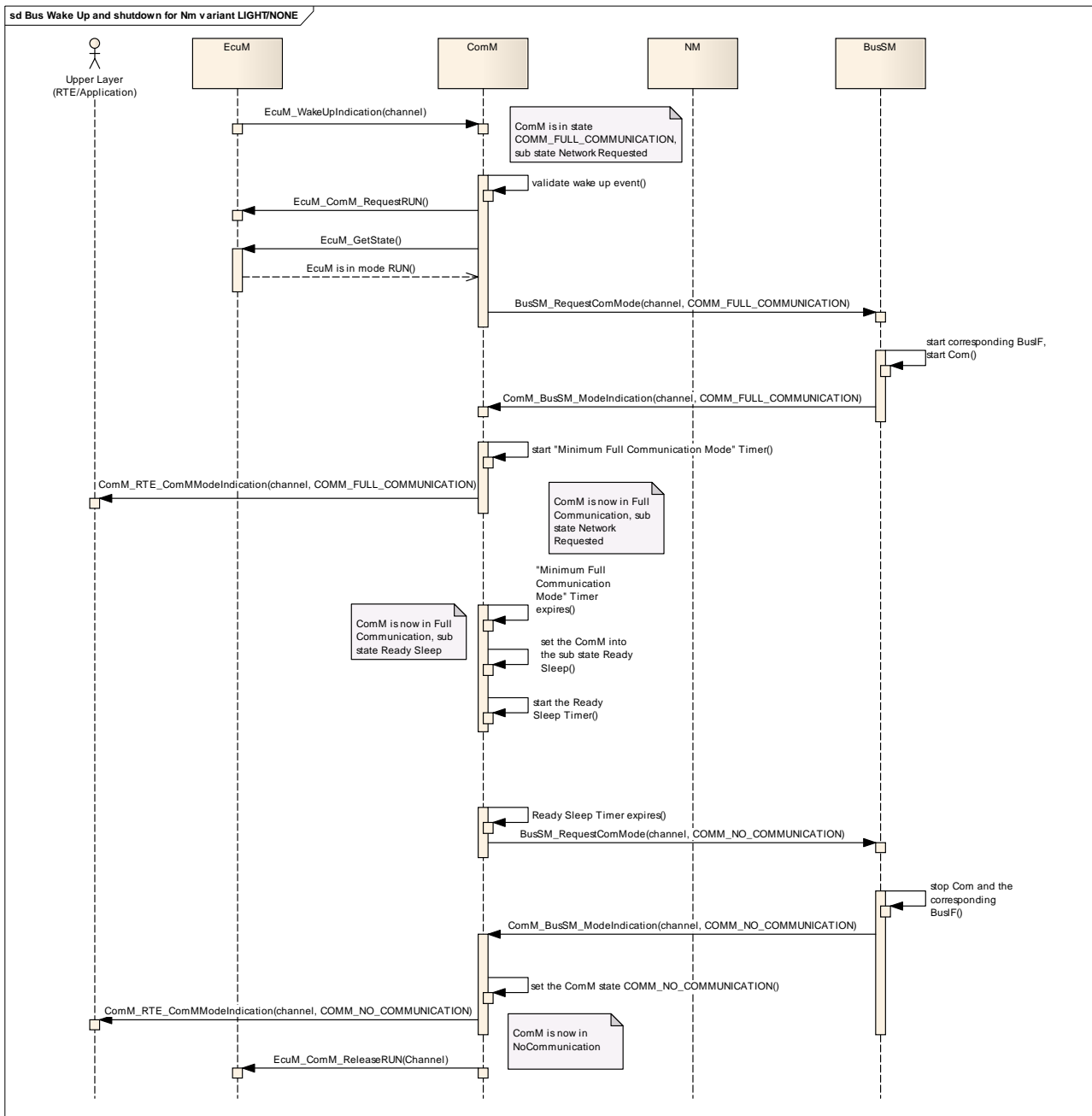
Figure 4-4 Bus Wake Up and shutdown for Nm variant LIGHT/NONE

## 4.4.2 Mode Limitation

Mode limitation is a mechanism to restrict the actions of the ComM user, specially 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 ECU's which are wrongly keeps awake the busses.

## 'No Communication' mode limitation

This mechanism can be used to force ComM channel(s) into the sleep mode although one or more ComM user requests FULL communication.

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

If the limitation is active than the ComM ignores all new communication mode requests for FULL communication, triggers the shutdown of the communication channels, and clear all FULL communication user requests if the ComM enters the state NO communication. Additionally it is configurable that the ComM triggers an ECU reset if the limitation is active and the ComM reaches the state NO communication.
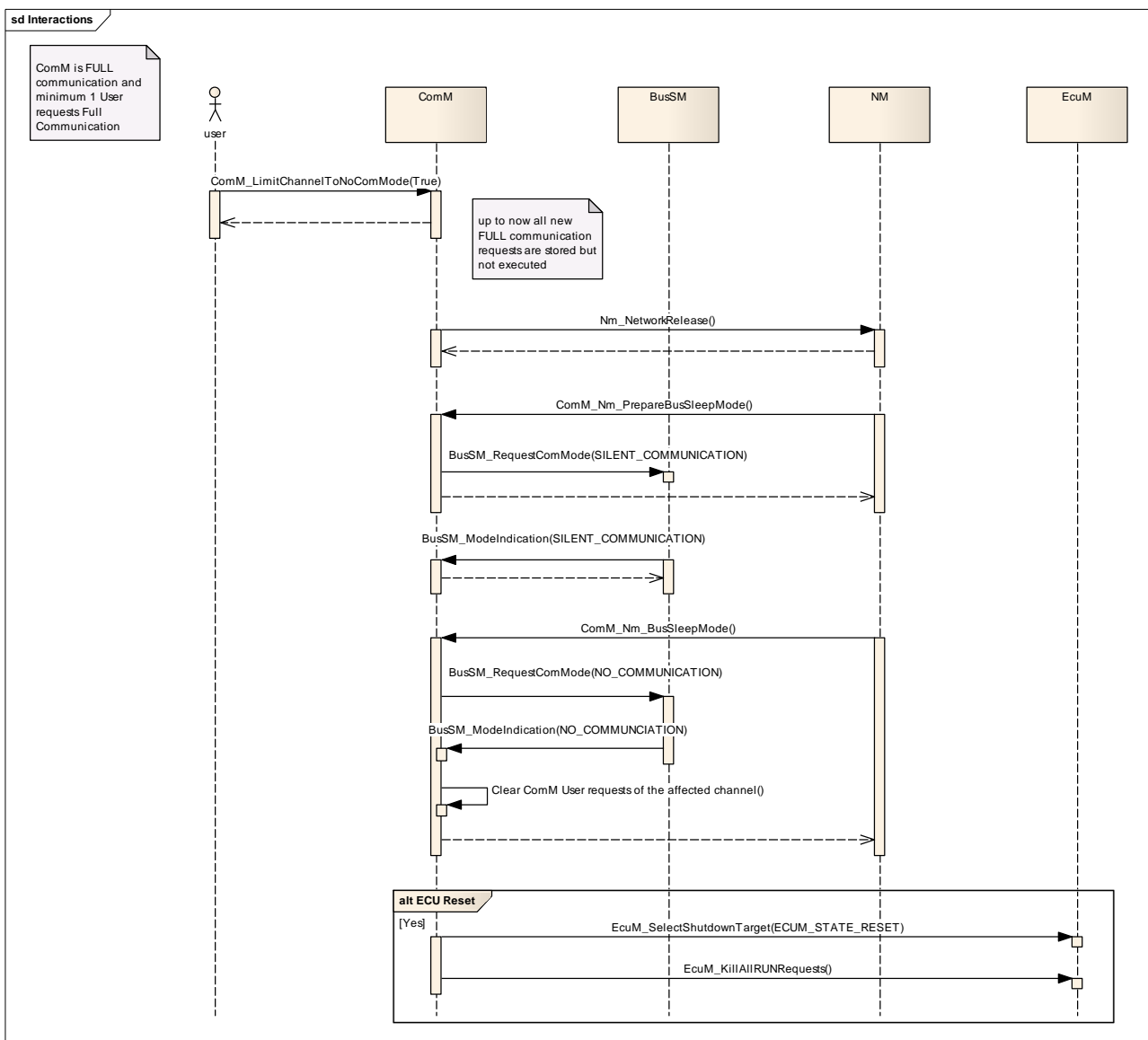


Figure 4-5 No Communication mode limitation

**Prevent Wake UP**

Prevent Wake Up is the second mode limitation mechanism; it avoids that ComM channels can be woken up via a FULL communication request via a ComM user. Prevent wake up can be activated/deactivated via ComM_PreventWakeUp() but the activation is only performed if the ComM is in state NO communication, user requests for FULL communication will be stored but not performed.

| | **Info** |
|---|---|
| **i** | The prevent wake up state is stored non-volatile. |

### 4.4.3 Synchronous Wake Up

Synchronous wake up means that the ComM triggers a wake up of all ComM busses if there notify an external wake up, e.g. via EcuM Wake up notification or via the notification of the configured NM.

| | **Caution** |
|---|---|
| ⚠ | Synchronous wake up does only trigger the wake up but is not responsible to keep awake all busses or responsible for a synchronous shutdown of these busses. |

### 4.4.4 Mode Indication

The ComM informs via the specified mode indication API the higher layered SW-Cs about the ComM state changes. It is possible to use the mode indication via the Rte mode management feature or without Rte via callback functions (refer to 6.6.4).

The mode indications are assigned and notified via the configured ComM user (for configuration refers to 7.1.4).

| | **Info** |
|---|---|
| **i** | ■ The ComM mode changes are informed always out of the ComM_MainFunction_x(). |
| | ■ The ComM does not queue the mode changes between the ComM mainfunctions, this means always the current mode is reported, if there is the last mode is different to the current mode. |
| | ■ If a ComM user assigned to more than one channel than the mode change notifies the lowest ComM state of all assigned channels. |

> **Example**
>
> ComM User: UR000, assigned to channel 0
>
> ComM User: UR001, assigned to channel 0 and channel 1
>
> Scenario 1:
>
> ComM changes his mode from NO to FULL communication on channel 0
>
> → mode indication for user UR000 is called
>
> → mode indication for user UR001 is not called because the second channel is in mode NO communication
>
> Scenario 2:
>
> ComM changes the mode for Channel 0 and channel 1 to FULL communication
>
> → mode indication for user UR000 and user UR0001 is called

## 4.5 Error Handling

### 4.5.1 Development Error Reporting

Development errors are reported to DET using the service Det_ReportError(), (specified in [2]), if the 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 6.3. 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_LimitToNoComMode |
| 0x0c | ComM_LimitECUToNoComMode |
| 0x0d | ComM_ReadInhibitCounter |
| 0x0e | ComM_ResetInhibitCounter |
| 0x0f | ComM_SetECUGroupClassification |
| 0x10 | ComM_GetVersionInfo |
| 0x60 | ComM_MainFunction |

Table 4-3   Mapping of service IDs to services

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 try to perform a not allowed state change. |

Table 4-4   Errors reported to DET

### 4.5.1.1 Parameter Checking

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

| Service | COMM_E_NOT_INITED | COMM_E_WRONG_PARAMETERS | COMM_E_ERROR_IN_PROVIDED_SERVICE | COMM_E_NOTSUPPORTED_MODECHANGE |
|---|---|---|---|---|
| ComM_Init | | | | |
| ComM_DeInit | | | | |
| ComM_GetStaus | | ■ | | |
| ComM_GetInhibitionStatus | ■ | ■ | | |
| ComM_RequestComMode | ■ | ■ | | |
| ComM_GetMaxComMode | ■ | ■ | | |
| ComM_GetRequestedComMode | ■ | ■ | | |
| ComM_GetCurrentComMode | ■ | ■ | | |
| ComM_PreventWakeUp | ■ | ■ | | |
| ComM_LimitChannelToNoComMode | ■ | ■ | | |
| ComM_LimitECUToNoComMode | ■ | ■ | | |
| ComM_ReadInhibitCounter | ■ | ■ | | |
| ComM_ResetInhibitCounter | ■ | | | |
| ComM_SetECUGroupClassification | ■ | ■ | | |
| ComM_GetVersionInfo | | ■ | | |
| ComM_MainFunction | ■ | ■ | ■ | ■ |

Table 4-5   Development Error Reporting: Assignment of checks to services

These checks are executed when COMM_DEV_ERROR_DETECT == STD_ON, which is enabled or disabled in the generation tool (refer to chapter 7.1.2).

### 4.5.2 Production Code Error Reporting

Production code related errors are reported to DEM using the service Dem_ReportErrorStatus() (specified in [3]).

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

| Error Code | Description |
|---|---|
| COMM_E_NET_START_IND_CHANNEL_<X><br><br>X being number of the channel e.g. 0 | Reception of a NM message in ComM state "No Communication" |

Table 4-6   Errors reported to DEM

# 5 Integration

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

## 5.1 Scope of Delivery

The delivery of the ComM contains the files which are described in the chapters 5.1.1 and 5.1.2:

### 5.1.1 Static Files

| File Name | 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_EcuM.h | Header File which includes the external declarations of the EcuM 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. |
| ComM.lib | This is the library file of the ComM. (optional) |

Table 5-1    Static files

### 5.1.2 Dynamic Files

The dynamic files are generated by the configuration tool GENy.

| 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_GenDataTypes.h | This file contains the generated type definitions of the ComM. |

Table 5-2    Generated files

## 5.2 Include Structure
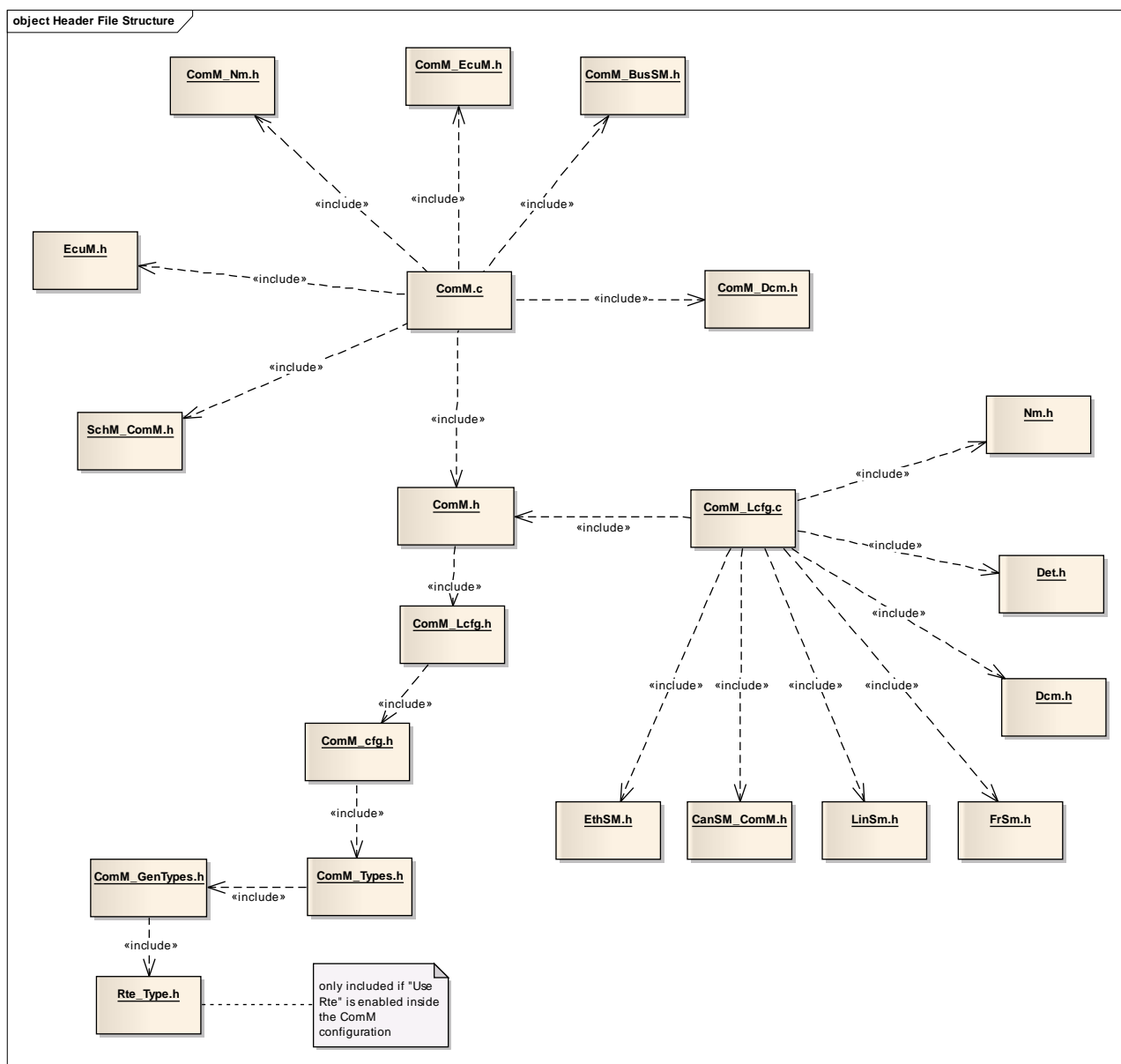


Figure 5-1 Include structure

## 5.3 Compiler Abstraction and Memory Mapping

The objects (e.g. variables, functions, constants) are declared by compiler independent definitions – the compiler abstraction definitions. Each compiler abstraction definition is assigned to a memory section.

The following table contains the memory section names and the compiler abstraction definitions that are used by the ComM. It illustrates their assignment among each other.

| Memory Mapping Sections | Compiler Abstraction Definitions | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | COMM_CONST | COMM_VAR_ZERO_INIT | COMM_CODE | COMM_APPL_VAR | COMM_VAR_NOINIT_8BIT | COMM_VAR_NOINIT_16BIT | COMM_VAR_NOINIT_UNSPECIFIED | COMM_VAR_ZERO_INIT | COMM_APPL_VAR_NVRAM |
| COMM_START_SEC_CONST_8BIT<br>COMM _STOP_SEC_CONST_8BIT | ■ | | | | | | | | |
| COMM_START_SEC_CONST_UNSPECIFIED<br>COMM_STOP_SEC_CONST_UNSPECIFIED | ■ | | | | | | | | |
| COMM_START_SEC_CODE<br>COMM_STOP_SEC_CODE | | | ■ | ■ | | | | | |
| COMM_START_SEC_VAR_NOINIT_8BIT<br>COMM_STOP_SEC_VAR_NOINIT_8BIT | | | | | ■ | | | | |
| COMM_START_SEC_VAR_NOINIT_16BIT<br>COMM_STOP_SEC_VAR_ NOINIT_16BIT | | | | | | ■ | | | |
| COMM_START_SEC_VAR_NOINIT_UNSPECIFIED<br>COMM_STOP_SEC_VAR_NOINIT_UNSPECIFIED | | | | | | | ■ | | |
| COMM_START_SEC_VAR_ ZERO_INIT_UNSPECIFIED<br>COMM_STOP_SEC_VAR_ ZERO_INIT_UNSPECIFIED | | | | | | | | ■ | |
| COMM_START_SEC_VAR_SAVED_ZONE0_UNSPECIFIED<br>COMM_STOP_SEC_VAR_SAVED_ZONE0_UNSPECIFIED | | | | | | | | | ■ |

Table 5-3   Compiler Abstraction and Memory Mapping

## 5.4   Critical code sections

The ComM has the following defined critical code sections:

- COMM_EXCLUSIVE_AREA_0: must lock task interrupts and interrupts sources

- COMM_EXCLUSIVE_AREA_1: must lock task interrupts, so that the ComM_MainFunction can not be interrupted by the following BSW Module tasks:

    o  Nm_MainFunction()

    o  CanNm_MainFunction()

    o  FrNm_MainFunction()

    o  CanSM_MainFunction()

    o  FrSM_MainFunction()

      o   LinSM_MainFunction()

It is recommended to use AUTOSAR OS 'Resources' for these exclusive areas to prevent priority inversions and dead-locks.

## 5.5 Handling of non-volatile data

The non-volatile data are 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 struct called `ComM_Inhibition`. The struct contains the following elements (order of elements equal to the struct 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 then the ComM works with the configured values.

> **Info**
> The non-volatile data handling is only necessary if the ComM configuration option 'Mode Limitation' is enabled.

# 6 API Description

## 6.1 Interfaces Overview
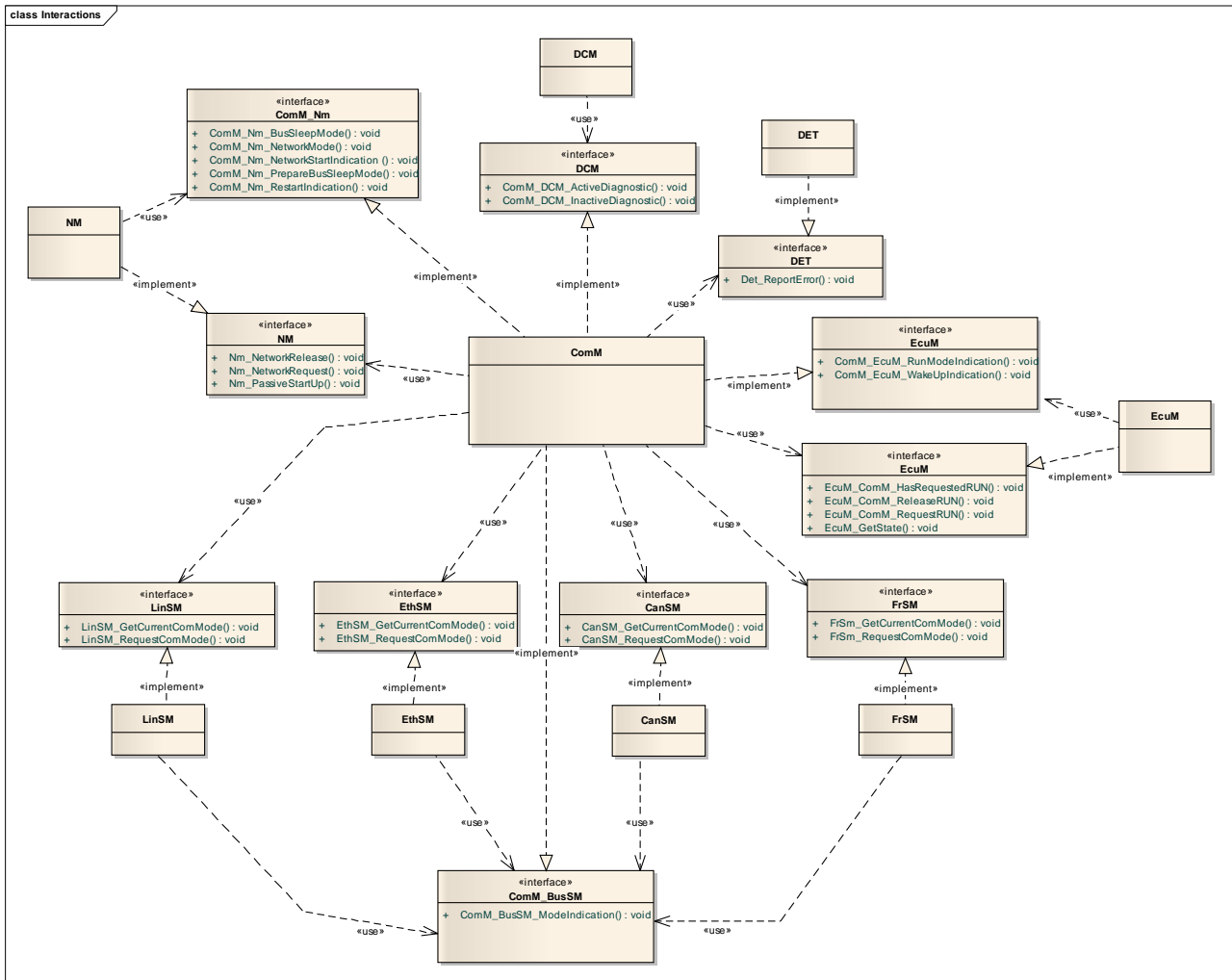


Figure 6-1 ComM interactions with other BSW

## 6.2   Type Definitions

| Type Name | C-Type | Description | Value Range |
|---|---|---|---|
| ComM_UserHandleType | uint8 | Defines the type of the ComM communication user | `0..255` |
| ComM_ModeType | uint8 | Defines the ComM main states | `COMM_NO_COMMUNICATION`<br>ComM is in the state No Communication |
| | | | `COMM_SILENT_COMMUNICATION`<br>ComM is in state Silent Communication |
| | | | `COMM_FULL_COMMUNICATION`<br>ComM is in state Full Communication |
| ComM_InitStatusType | enum | The enum contains the information | `COMM_UNINIT`<br>ComM is not initialized |
| | | | `COMM_INIT`<br>ComM is initialized |
| ComM_InhibitionType | struct | contains the ComM non-volatile data elements | `ComM_InhibitionStatusType ComM_ECUGroupClassification`<br>ECU group classification |
| | | | `uint16 ComM_InhibitCnt`<br>ComM Inhibition counter |
| | | | `ComM_InhibitionStatusType ComM_InhibitionStatus[<COMM_ACTIVE_CHANNEL>]`<br>ComM Inhibitions state per ComM channel |

Table 6-1   Type definitions

## 6.3 Services provided by ComM

The ComM API consists of services, which are realized by function calls.

### 6.3.1 ComM_InitMemory

| Prototype | |
|---|---|
| void **ComM_InitMemory** ( void ) | |
| **Parameter** | |
| none | - |
| **Return code** | |
| none | - |
| **Functional Description** | |
| This function initializes the ComM memory and set the init state to COMM_UNINT. | |
| **Particularities and Limitations** | |
| ■  This function is to be called only once during PowerOn. | |
| Expected Caller Context | |
| ■  - | |

Table 6-2   ComM_InitMemory

### 6.3.2 ComM_Init

| Prototype | |
|---|---|
| Multiple identity configuration disabled: | |
| void **ComM_Init** ( void ) | |
| Multiple identity configuration enabled: | |
| void **ComM_Init** ( const ComM_ConfigSetType* ConfigPtr ) | |
| **Parameter** | |
| ConfigPtr | Pointer to the ComM configuration. Is only used for multiple identity configurations, in this case ConfigPtr contains the information which identity is enabled. This information is provided by the generated file ComM_Lcfg.c via the variables of type ComM_ConfigSetType which have the name ComM_<MultipleConfigurationContainerName>. |
| **Return code** | |
| none | - |
| **Functional Description** | |
| This function initializes the ComM. All variables are set to default values. The ComM init state is set to COMM_INIT and the ComM main state is set to COMM_NO_COMMUNICATION. | |
| For multiple identity configurations the ComM stores the current channel to identity mapping for further processing, means the ComM performs only actions for channels which are active in the current identity. | |
| **Particularities and Limitations** | |
| ■  This function is to be called only once during PowerOn after ComM_InitMemory was called. | |
| ■  If 'Mode Limitation' is enabled than the non-volatile values must be loaded and stored before this function can be called (see 5.5). | |

| Expected Caller Context |
|---|
| ■ - |

Table 6-3   ComM_Init

### 6.3.3   ComM_GetStatus

| Prototype |
|---|
| Std_ReturnType **ComM_GetStatus** ( ComM_InitStatusType* Status ) |

| Parameter | |
|---|---|
| Status | Pointer where the ComM init status shall be stored |

| Return code | |
|---|---|
| E_OK | ■ ComM_GetStatus has performed |
| E_NOT_OK | ■ ComM is not initialized |
| | ■ Status is a NULL pointer |

| Functional Description |
|---|
| This function gets the initialization status of the ComM. |

| Particularities and Limitations |
|---|
| ■ - |

| Expected Caller Context |
|---|
| ■ Function can be called in task and interrupt context |

Table 6-4   ComM_GetStatus

### 6.3.4   ComM_GetInhibitionStatus

| Prototype |
|---|
| Std_ReturnType **ComM_GetInhibitionStatus** ( NetworkHandleType Channel, ComM_InihibitionStatusType* Status ) |

| Parameter | |
|---|---|
| Channel | Index of the system channel |
| Status | Pointer where the ComM inihibition status shall be stored |

| Return code | |
|---|---|
| E_OK | ■ ComM_GetInhibitionStatus has performed |
| E_NOT_OK | ■ not valid parameter |
| COMM_E_UNINIT | ■ ComM is not initialized |

| Functional Description |
|---|
| This function gets the current inhibition status of the ComM. |

| Particularities and Limitations |
|---|
| ■ - |

| Expected Caller Context |
|---|
| ■ Function can be called in task and interrupt context |

Table 6-5   ComM_GetInhibitionStatus

### 6.3.5 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_Lcfg.h file |
| ComMode | The requested communication mode:<br>■ COMM_FULL_COMMUNICATION<br>■ COMM_NO_COMMUNICATION |
| **Return code** | |
| E_OK | ■ Request is accepted |
| E_NOT_OK | ■ not valid parameter |
| COMM_E_UNINIT | ■ ComM is not initialized |
| COMM_E_MODE_LIMITATION | ■ Requested was successful but mode can not 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 proceed in the ComM_MainFunction(). | |
| **Particularities and Limitations** | |
| ■  - | |
| Expected Caller Context | |
| ■ Function can be called in task and interrupt context | |

Table 6-6    ComM_RequestComMode

### 6.3.6 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_Lcfg.h file |
| ComMode | Pointer where the maximal communication mode of the given use shall be stored |
| **Return code** | |
| E_OK | ■ Request is accepted |
| E_NOT_OK | ■ not valid 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** | |
| ■  - | |

| Expected Caller Context |
|---|
| ■ Function can be called in task and interrupt context |

Table 6-7   ComM_GetMaxComMode

## 6.3.7   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_Lcfg.h file |
| ComMode | Pointer where the requested communication mode of the given use shall be stored |
| **Return code** | |
| E_OK | ■ Request is accepted |
| E_NOT_OK | ■ not valid parameter |
| COMM_E_UNINIT | ■ ComM is not initialized |
| **Functional Description** | |
| This function queries the requested communication mode of the corresponding user. | |
| **Particularities and Limitations** | |
| ■ - | |
| Expected Caller Context | |
| ■ Function can be called in task and interrupt context | |

Table 6-8   ComM_GetRequestedComMode

### 6.3.8    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_Lcfg.h file |
| ComMode | Pointer where the current communication mode of the given use shall be stored |
| **Return code** | |
| E_OK | ■   Request is accepted |
| E_NOT_OK | ■   not valid 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** | |
| ■   - | |
| Expected Caller Context | |
| ■   Function can be called in task and interrupt context | |

Table 6-9   ComM_GetCurrentComMode

### 6.3.9    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<br>■   FALSE: Wake Up Inhibition is switched off |
| **Return code** | |
| E_OK | ■   Request is accepted |
| E_NOT_OK | ■   not valid parameter |
| 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** | |
| ■   - | |
| Expected Caller Context | |
| ■   Function can be called in task and interrupt context | |

Table 6-10 ComM_PreventWakeUp

### 6.3.10 ComM_LimitChannelToNoComMode

| Prototype | |
|---|---|
| `Std_ReturnType` **`ComM_LimitChannelToNoComMode`** `( NetworkHandleType Channel, boolean Status )` | |
| **Parameter** | |
| Channel | Index of the system channel |
| Status | ■ TRUE: limitation to COMM_NO_COMMUNCIATION is ON<br>■ FALSE: limitation to COMM_NO_COMMUNCIATION is OFF |
| **Return code** | |
| E_OK | ■ Request is accepted |
| E_NOT_OK | ■ not valid parameter |
| COMM_E_UNINIT | ■ ComM is not initialized |
| **Functional Description** | |
| This function set the maximal communication mode of the given channel to COMM_NO_COMMUNICATION. | |
| **Particularities and Limitations** | |
| ■  - | |
| Expected Caller Context | |
| ■ Function can be called in task and interrupt context | |

Table 6-11 ComM_LimitChannelToNoComMode

### 6.3.11 ComM_LimitECUToNoComMode

| Prototype | |
|---|---|
| `Std_ReturnType` **`ComM_LimitECUToNoComMode`** `( boolean Status )` | |
| **Parameter** | |
| Status | ■ TRUE: limitation to COMM_NO_COMMUNCIATION is ON<br>■ FALSE: limitation to COMM_NO_COMMUNCIATION is OFF |
| **Return code** | |
| E_OK | ■ Request is accepted |
| E_NOT_OK | ■ not valid parameter |
| COMM_E_UNINIT | ■ ComM is not initialized |
| **Functional Description** | |
| This function set the maximal communication mode of all configured ComM channels of the ECU to COMM_NO_COMMUNICATION. | |
| **Particularities and Limitations** | |
| ■  - | |
| Expected Caller Context | |
| ■ Function can be called in task and interrupt context | |

Table 6-12 ComM_LimitECUToNoComMode

### 6.3.12 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 | ■ not valid parameter |
| COMM_E_UNINIT | ■ ComM is not initialized |
| **Functional Description** | |
| This function returns the amount of rejected "Full Communciation" user requests. | |
| **Particularities and Limitations** | |
| ■   - | |
| Expected Caller Context | |
| ■   Function can be called in task and interrupt context | |

Table 6-13 ComM_ReadInhibitCounter

### 6.3.13 ComM_ResetInhibitCounter

| Prototype | |
|---|---|
| Std_ReturnType **ComM_ReadInhibitCounter** ( void  ) | |
| **Parameter** | |
| none | - |
| **Return code** | |
| E_OK | ■ Request is accepted |
| COMM_E_UNINIT | ■ ComM is not initialized |
| **Functional Description** | |
| This function reset the counter of rejected "Full Communciation" user requests. | |
| **Particularities and Limitations** | |
| ■   - | |
| Expected Caller Context | |
| ■   Function can be called in task and interrupt context | |

Table 6-14 ComM_ResetInhibitCounter

based on template version 3.1

### 6.3.14 ComM_SetECUGroupClassification

| Prototype | |
|---|---|
| Std_ReturnType **ComM_SetECUGroupClassification** ( ComM_InhibitionStatusType Status ) | |
| **Parameter** | |
| Status | Defines whether if the prevent wake up or the mode limitation to COMM_NO_COMMUNCIATION have effects for the ECU |
| **Return code** | |
| E_OK | ■ Request is accepted |
| E_NOT_OK | ■ not valid parameter |
| COMM_E_UNINIT | ■ ComM is not initialized |
| **Functional Description** | |
| This function changes the ECU group classification status during runtime. The ECU group classification defines if the ECU is affected by a bus wake up inhibition or by a ComM mode limitation to COMM_NO_COMMUNICATION. | |
| **Particularities and Limitations** | |
| ■ - | |
| Expected Caller Context | |
| ■ Function can be called in task and interrupt context | |

Table 6-15 ComM_SetECUGroupClassification

### 6.3.15 ComM_GetVersionInfo

| Prototype | |
|---|---|
| void **ComM_GetVersionInfo** ( Std_versionInfoType* versioninfo ) | |
| **Parameter** | |
| versioninfo | Pointer where the version information shall be stored. |
| **Return code** | |
| none | - |
| **Functional Description** | |
| This function is called to get the version information of the ComM. | |
| **Particularities and Limitations** | |
| ■ 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 6-16 ComM_GetVersionInfo

### 6.3.16 ComM_MainFunction

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

| Parameter | |
|---|---|
| none | - |

| Return code | |
|---|---|
| none | - |

| Functional Description |
|---|
| This function must be called cyclically with the configured ComM cycle time (refer to chapter 7.1.2). The ComM performs inside this function the channel specific state transitions. |
| For multiple identity configurations the ComM only executes channel specific main functions for channels which are active in the current identity. |

| Particularities and Limitations |
|---|
| ■  - |

| Expected Caller Context |
|---|
| ■  Function must be called in task and not in a reentrant way |

Table 6-17 ComM_MainFunction

## 6.4 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 |
| DEM | Dem_SetEventStatus |
| CanSM | CanSM_RequestComMode |
| CanSM | CanSM_GetCurrentComMode |
| LinSM | LinSM_RequestComMode |
| LinSM | LinSM_GetCurrentComMode |
| FrSM | FrSm_RequestComMode |
| FrSM | FrSm_GetCurrentComMode |
| EthSM | EthSM_RequestComMode |
| EthSM | EthSM_GetCurrentComMode |
| NmIf | Nm_PassiveStartUp |
| NmIf | Nm_NetworkRequest |
| NmIf | Nm_NetworkRelease |
| EcuM | EcuM_ComM_RequestRUN |
| EcuM | EcuM_ComM_ReleaseRUN |
| EcuM | EcuM_ComM_HasRequestedRUN |
| EcuM | EcuM_GetState |
| EcuM | EcuM_SetSelectShutdownTarget |
| EcuM | EcuM_KillAllRUNRequests |
| EcuM | EcuM_GeneratorCompatibilityError refer to [12] |

Table 6-18 Services used by the ComM

## 6.5 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_Nm.h, ComM_BusSM.h, ComM_Dcm.h and ComM_EcuM.h by the ComM.

### 6.5.1 ComM_EcuM_RunModeIndication

| Prototype | |
|---|---|
| void **ComM_EcuM_RunModeIndication** ( const NetworkHandleType Channel ) | |
| **Parameter** | |
| Channel | Index of the system channel |
| **Return code** | |
| None | - |
| **Functional Description** | |
| This function notifies the ComM that the EcuM has entered the RUN mode. | |
| **Particularities and Limitations** | |
| ■  - | |
| Expected Caller Context | |
| ■  - | |

Table 6-19 ComM_EcuM_RunModeIndication

### 6.5.2 ComM_EcuM_WakeUpIndication

| Prototype | |
|---|---|
| void **ComM_EcuM_WakeUpIndication** ( const NetworkHandleType Channel ) | |
| **Parameter** | |
| Channel | Index of the system channel |
| **Return code** | |
| None | - |
| **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. | |
| **Particularities and Limitations** | |
| ■  - | |
| Expected Caller Context | |
| ■  - | |

Table 6-20 ComM_EcuM_WakeUpIndication

### 6.5.3 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 | |
|---|---|
| None | - |

| 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 |
|---|
| ■  - |

| Expected Caller Context |
|---|
| ■  - |

### 6.5.5  ComM_DCM_InactiveDiagnostic

| Prototype | |
|---|---|
| void **ComM_DCM_InactiveDiagnostic** ( void ) | |
| **Parameter** | |
| None | - |
| **Return code** | |
| None | - |
| **Functional Description** | |
| This function notifies the ComM about the end of the DCM diagnostic session. The ComM triggers the network shutdown of all channels, if all ComM user requests the ComM state No Communciation | |
| **Particularities and Limitations** | |
| ■  - | |
| Expected Caller Context | |
| ■  - | |

Table 6-23 ComM_DCM_IncactiveDiagnostic

### 6.5.6  ComM_Nm_NetworkStartIndication

| Prototype | |
|---|---|
| void **ComM_Nm_NetworkStartIndication** ( const NetworkHandleType Channel ) | |
| **Parameter** | |
| Channel | Index of the system channel |
| **Return code** | |
| None | - |
| **Functional Description** | |
| This function notifies the ComM about a restart of the network management. The restart was triggered by receiving an Nm message. The ComM stores the event and starts up the corresponding network. | |
| **Particularities and Limitations** | |
| ■  If the Production Error support is enabled in the ComM configuration then the ComM reports the error COMM_E_NET_START_IND_CHANNEL_x to the DEM.  (x being number of channel e.g. 0) | |
| Expected Caller Context | |
| ■  - | |

Table 6-24 ComM_Nm_NetworkStartIndication

### 6.5.7 ComM_Nm_NetworkMode

| Prototype |
|---|
| void **ComM_Nm_NetworkMode** ( const NetworkHandleType Channel ) |

| Parameter | |
|---|---|
| Channel | Index of the system channel |

| Return code | |
|---|---|
| None | - |

| Functional Description |
|---|
| This function notifies the ComM that the NM entered the state called "Network Mode". |

| Particularities and Limitations |
|---|
| ■  - |

| Expected Caller Context |
|---|
| ■  - |

Table 6-25 ComM_Nm_NetworkMode

### 6.5.8 ComM_Nm_PrepareBusSleep

| Prototype |
|---|
| void **ComM_Nm_PrepareBusSleep** ( const NetworkHandleType Channel ) |

| Parameter | |
|---|---|
| Channel | Index of the system channel |

| Return code | |
|---|---|
| None | - |

| Functional Description |
|---|
| This function notifies the ComM that the NM starts the prepare bus sleep phase. 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 self changes into the mode Silent Communication. |

| Particularities and Limitations |
|---|
| ■  - |

| Expected Caller Context |
|---|
| ■  - |

Table 6-26 ComM_Nm_PrepareBusSleep

### 6.5.9   ComM_Nm_BusSleepMode

| Prototype | |
| --- | --- |
| `void `**`ComM_Nm_BusSleep`**` ( const NetworkHandleType Channel )` | |
| **Parameter** | |
| Channel | Index of the system channel |
| **Return code** | |
| None | - |
| **Functional Description** | |
| This function notifies the ComM that the NM ends the prepare bus sleep phase and changes his state into sleep. 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 self changes into the mode no Communication. | |
| **Particularities and Limitations** | |
| ■   - | |
| Expected Caller Context | |
| ■   - | |

Table 6-27 ComM_Nm_BusSleepMode

### 6.5.10  ComM_Nm_RestartIndication

| Prototype | |
| --- | --- |
| `void `**`ComM_Nm_RestartIndication`**` ( const NetworkHandleType Channel )` | |
| **Parameter** | |
| Channel | Index of the system channel |
| **Return code** | |
| None | - |
| **Functional Description** | |
| This function notifies the ComM that the NM has received a NM message. The Nm breaks the shutdown of the corresponding channel and restart the Nm. The ComM wakes up the network for the given channel. | |
| **Particularities and Limitations** | |
| ■   - | |
| Expected Caller Context | |
| ■   - | |

Table 6-28 ComM_Nm_RestartIndication

## 6.6   Configurable Interfaces

At  it

### 6.6.1    Dcm_ComM_FullComModeEntered

| Prototype | |
| --- | --- |
| void **Dcm_ComM_FullComModeEntered** ( void ) | |
| **Parameter** | |
| None | - |
| **Return code** | |
| None | - |
| **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 is the DCM is configured inside the system. | |
| Call Context | |
| ■  - | |

Table 6-29 Dcm_ComM_FullComModeEntered

### 6.6.2    Dcm_ComM_SilentComModeEntered

| Prototype | |
| --- | --- |
| void **Dcm_ComM_SilentComModeEntered** ( void ) | |
| **Parameter** | |
| None | - |
| **Return code** | |
| None | - |
| **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 is the DCM is configured inside the system. | |
| Call Context | |
| ■  - | |

Table 6-30 Dcm_ComM_SilentComModeEntered

### 6.6.3 Dcm_ComM_NoComModeEntered

| Prototype |
| --- |
| void **Dcm_ComM_NoComModeEntered** ( void ) |

| Parameter | |
| --- | --- |
| None | - |

| Return code | |
| --- | --- |
| None | - |

| 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 is the DCM is configured inside the system. |

| Call Context |
| --- |

■  -

| Call Context |
|---|
| ■ Task |

Table 6-32 Appl_ComM_<CurrentModePortPrefix><UserName>_currentMode

## 6.6.5 Rte_Switch_<CurrentModePortPrefix><UserName>_currentMode

| Prototype | |
|---|---|
| `Std_ReturnType` **`Rte_Switch_<CurrentModePortPrefix><UserName>_currentMode`** `(Rte_ModeType_ComMMode mode)` | |
| **Parameter** | |
| mode | ■ The parameter gets the new ComM mode: <br> ■ RTE_MODE_ComMMode_NO_COMMUNICATION, no communication is entered <br> ■ RTE_MODE_ComMMode_SILENT_COMMUNICATION, silent communication is entered <br> ■ RTE_MODE_ComMMode_FULL_COMMUNICATION, full communication is entered |
| **Return code** | |
| Std_ReturnType | ■ RTE_E_OK, the SW-C notified <br> ■ RTE_E_LIMIT, the SW-C does not notified the mode and the ComM shall informed again <br><br> ⚠ **Caution** <br> **The repeated mode indication is only called if the last mode is different to the current mode during the next call of the ComM_MainFunction_x().** |
| **Functional Description** | |
| This callback functions inform the SW-C about a ComM state change. | |
| **Particularities and Limitations** | |
| ■ This callback function is only available if Rte is used (configuration switch "Used Rte" == enabled). <br> ■ Only callback functions available for the ComM user where the Mode Notification is enabled. | |
| **Call Context** | |
| ■ Task | |

Table 6-33 Rte_Switch_<CurrentModePortPrefix><UserName>_currentMode

### 6.6.6 BswM_ComM_CurrentMode

| Prototype | |
|---|---|
| void **BswM_ComM_CurrentMode** (NetworkHandleType Network, ComM_ModeType RequestedMode) | |
| **Parameter** | |
| Network | Index of the system channel |
| Requested Mode | The current state where the ComM changed to. Refer to chapter 6.2 about of possible values. |
| **Return code** | |
| None | - |
| **Functional Description** | |
| The ComM calls this function every time when the state changes. | |
| **Particularities and Limitations** | |
| ■ This callback function is only available if the BswM is configured inside the system. | |
| Call Context | |
| ■ Is called in task context. | |

Table 6-34 BswM_ComM_CurrentMode

## 6.7 Service Ports

### 6.7.1 Client Server Interface

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

#### 6.7.1.1 Provide Ports

At the Provide Ports of the ComM the API functions described in 6.3 are available as Runable Entities. The Runable rfi Tdinvoknctvia Os described3ble Ce             Co

| Provide Port | Operation | API Function | Port Defined Argument Values |
|---|---|---|---|
| | GetRequestedComMode | ComM_GetRequestedComMode | ComM_UserHandleType UserHandle |
| ComM_ChannelWakeUp | PreventWakeUp | ComM_PrevetWakeUp | NetworkHandleType Channel |
| ComM_ChannelLimitation | LimitChannelToNoComMode | ComM_LimitChannelToNoComMode | NetworkHandleType Channel |
| | GetInhibitionStatus | ComM_GetInhibitionStatus | NetworkHandleType Channel |
| ComM_ECUModeLimitation | LimitECUToNoComMode | ComM_LimitECUToNoComMode | - |
| | ReadInhibitCounter | ComM_ReadInhibitCounter | - |
| | ResetInhibitCounter | ComM_ResetInhibtCounter | - |
| | SetECUGroupClassification | ComM_SetECUGroupClassification | - |

Table 6-35 Provide Ports on BSW module side

### 6.7.1.2 Require Ports

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

The following sub-chapters present the Require Ports defined for the ComM, the Operations that are called from the ComM and the related Notifications, which are described in chapter 6.6.5.

#### 6.7.1.2.1 Current Mode

| Operation | Rte Interface | Mode Declaration Group |
|---|---|---|
| currentMode | Rte_Switch_<CurrentModePortPrefix><UserName>_currentMode | ■ RTE_MODE_ComMMode_FULL_COMMUNICATION<br>■ RTE_MODE_ComMMode_SILENT_COMMUNICATION<br>■ RTE_MODE_ComMMode_NO_COMMUNICATION |

Table 6-36 Current Mode

# 7  Configuration

In the ComM the attributes can be configured with the following methods:

- Configuration in GENy for a detailed description see 7.1

## 7.1  Configuration with GENy

The ComM is configured with the help of the configuration tool GENy.

### 7.1.1  Activation of the ComM in GENy

The component name of the Communication Manager in GENy is Ccl_AsrComM. In the Component Selection page the ComM can be enabled on all communication channels.
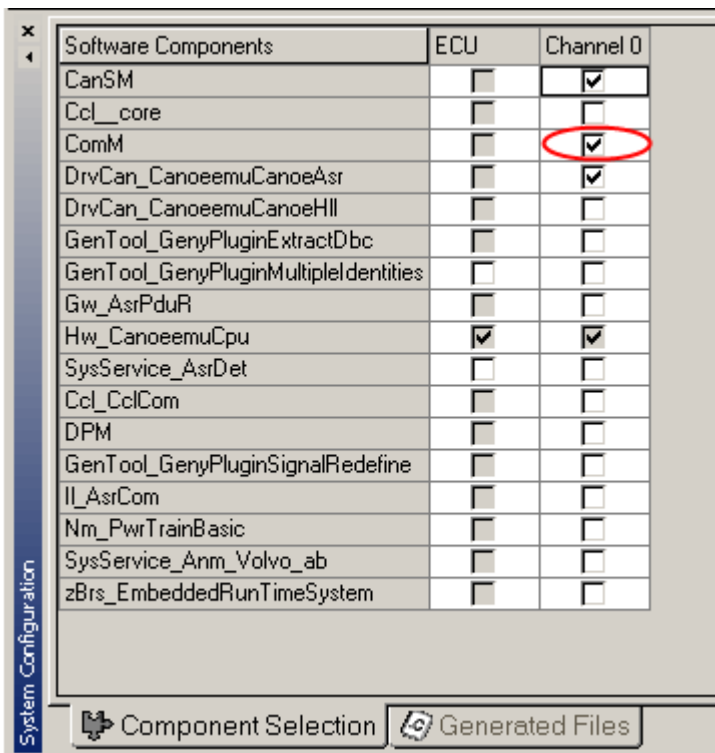


Figure 7-1  GENy Component Selection Dialog

## 7.1.2    General Configuration

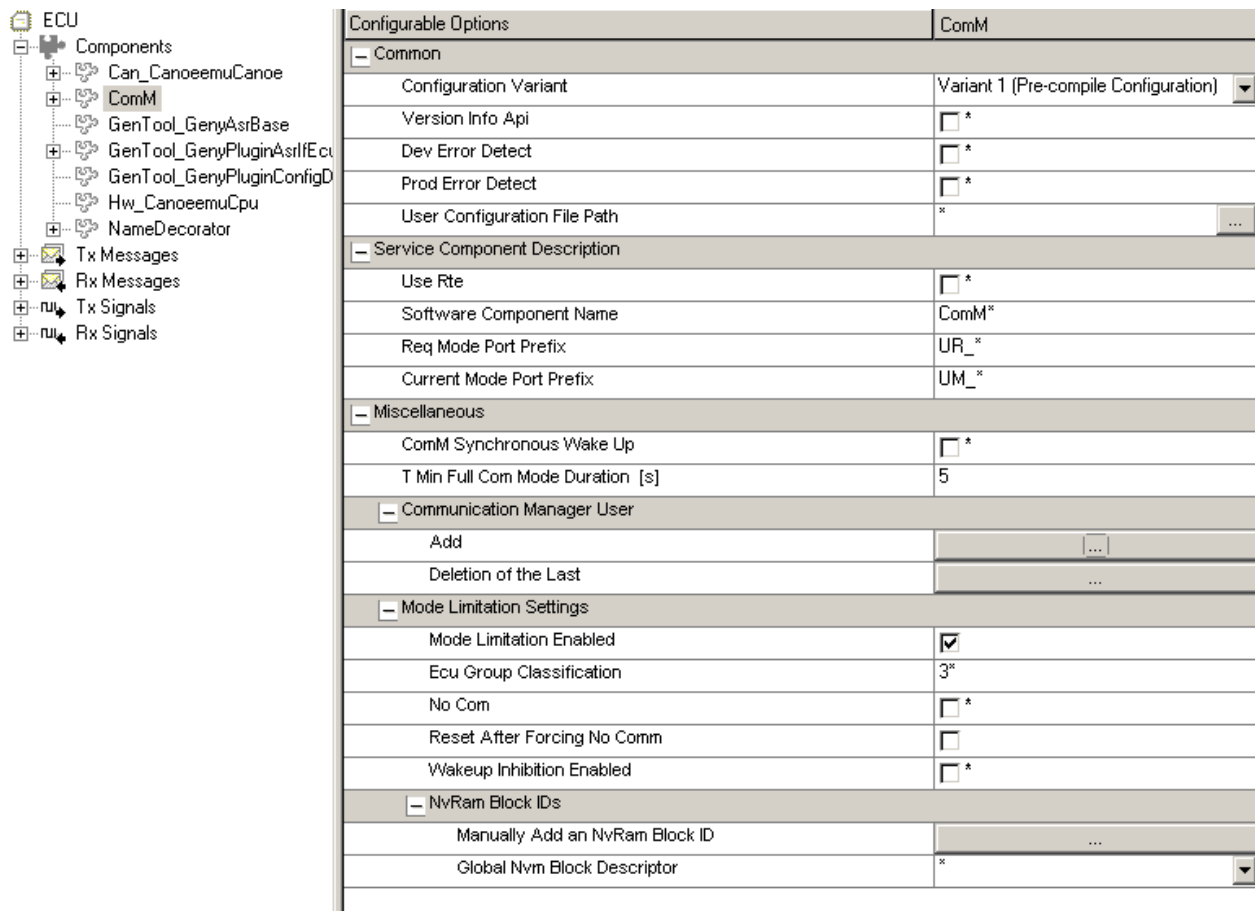This chapter describes the general configuration of the ComM.



Figure 7-2 General Configuration Settings in GENy

| Attribute Name | Configuration Variant | Value Type | Values<br>The default value is written in bold | Description |
|---|---|---|---|---|
| Common | | | | |
| Configuration Variant | - | - | Pre-Compile<br>Link-Time<br>Post-build | Specify the configuration variant |
| Version Info Api | pre-compile | boolean | ON<br>**OFF** | Enable/disable the function ComM_GetVersionInfo() to get the version information about the ComM. |
| Dev Error Detect | pre-compile | boolean | ON<br>**OFF** | If 'Development Error Detection' is ON, development errors are reported to the Development Error Tracer (Det). |
| Prod Error Detect | pre-compile | boolean | ON<br>**OFF** | If 'Production Error Detection' is enabled, production relevant errors are reported to the Diagnostics Event Manager |

based on template version 3.1

| Attribute Name | Configuration Variant | Value Type | Values<br>The default value is written in bold | Description |
|---|---|---|---|---|
| | | | | (DEM).<br>Disable this option, if no DEM is present in your system. |
| User Configuration File Path | pre-compile | String | | A ComM configuration file is generated by GENy (ComM_cfg.h). If you want to overwrite settings in the generated file, you can specify a path to a user defined configuration file. The user defined configuration file will be included at the end of the generated file. Therefore definitions in the user defined configuration file can overwrite definitions in the generated configuration file. |
| Service Component Description | | | | |
| Use Rte | link-time | boolean | ON<br>**OFF** | Enable this option if you use a Rte.<br>If this option is enabled, a port interface description (Software Component Template) is generated which can be used for the Rte configuration. |
| Software Component Name | pre-compile | String | **ComM**<br>or any ANSI-C string | This name is used as component name (ShortName) in the generated software component template. Change this name if you try to import the ComM components of several ECUs and have problems with name clashes. |
| Req Mode Port Prefix | link-time | String | **UR_**<br>or any ANSI-C string | Enter the prefix of the ComM_RequestMode service port. The prefix and the symbolic name of the ComM user build the name of the service port, e.g.<br>Prefix = UR_<br>User Handle Name = RunableUser_1<br><br>Service Port Name = UR_RunableUser_1 |
| Current Mode Port Prefix | Link-time | String | **UM_**<br>or any ANSI-C string | Enter the prefix of the ComM_CurrentMode service port. The prefix and the symbolic name of the ComM user build the name of the service port, e.g.<br>Prefix = UM_<br>User Handle Name = RunableUser_1 |

| Attribute Name | Configuration Variant | Value Type | Values<br>The default value is written in bold | Description |
|---|---|---|---|---|
| | | | | Service Port Name = UM_RunableUser_1 |
| Miscellaneous | | | | |
| ComM Synchronous Wake Up | link-time | boolean | ON<br>**OFF** | Enable the synchronous wake up of the ComM busses. |
| T Min Full Com Mode Duration [s] | link-time | Float | 0.0010..65.000 | Minimum time duration in seconds, spent in full communication mode (only available for Nm variant LIGHT, NONE and OSEK) |
| Communication Manager User | | | | |
| Add | - | - | - | With this button you could create a further empty communication manager user. Click on the user settings to see the result. |
| Deletion of the Last | - | - | - | This button deletes the last communication manager user from the list. Currently it is only possible to delete the last user in the configuration view called 'User Settings'. |
| Mode Limitation Settings | | | | |
| Mode Limitation Enabled | Link-time | boolean | ON<br>**OFF** | Enable/Disable the ComM mode limitation functionality:<br>■ Bus Wake Up Inhibition<br>■ Mode limitation to COMM_NO_COMMUNICATION |
| Ecu Group Classification | Link-Time | int | ■ 0<br>■ 1<br>■ 2<br>■ 3 | Defines the initialization value if the ECU is affected by a bus wake up inhibition or by a mode limitation to COMM_NO_COMMUNICATION<br>■ 0, not affected by any limitation<br>■ 1, only affected by bus wake up inhibition<br>■ 2, only affected by mode limitation to COMM_NO_COMMUNICATION<br>■ 3, is affected by both described limitations<br>■ This value can be changed during runtime vie the API ComM_SetECUGroupClassification() |
| No Com | Link-Time | boolean | ON<br>**OFF** | Defines the initialization value of the ECU mode limitation to COMM_NO_COMMUNICATION. |

| Attribute Name | Configuration Variant | Value Type | Values<br>The default value is written in bold | Description |
|---|---|---|---|---|
| | | | | This value can be changed during runtime via the API ComM_LimitECUToNoComMode(). |
| Reset After Forcing No Comm | Link-Time | boolean | ON<br>**OFF** | Defines if the ComM shall trigger an ECU reset after entering the ComM mode COMM_NO_COMMUNICATION in case of a mode limitation. |
| Wakeup Inhibition Enabled | Link-Time | boolean | ON<br>**OFF** | Defines the initialization value of the ECU bus wake up inhibition. |
| NvRam Block IDs | | | | |
| Manually Add an NvRam Block ID | - | - | - | This button adds a new Nvram Block ID that can then be selected for ComM.<br><br>This option is to configure a *nonexisting* NvRam BlockID for use with the ComM. You will still need to configure that NvRam BlockID in the NvRam manager! |
| Global Nvm Block Descriptor | Link-Time | String | Selection Box which contains the NvM block description definitions | Choose the block descriptor, which is used to handle the non-volatile data of the ComM.<br>If the selection box is empty then the ComM has found no NvM block descriptor definitions inside the ECU configuration file.<br>Please create a NvM Block description block for the ComM by using the button above. |

Table 7-1    General configuration Settings in GENy

### 7.1.3    Channel Configuration
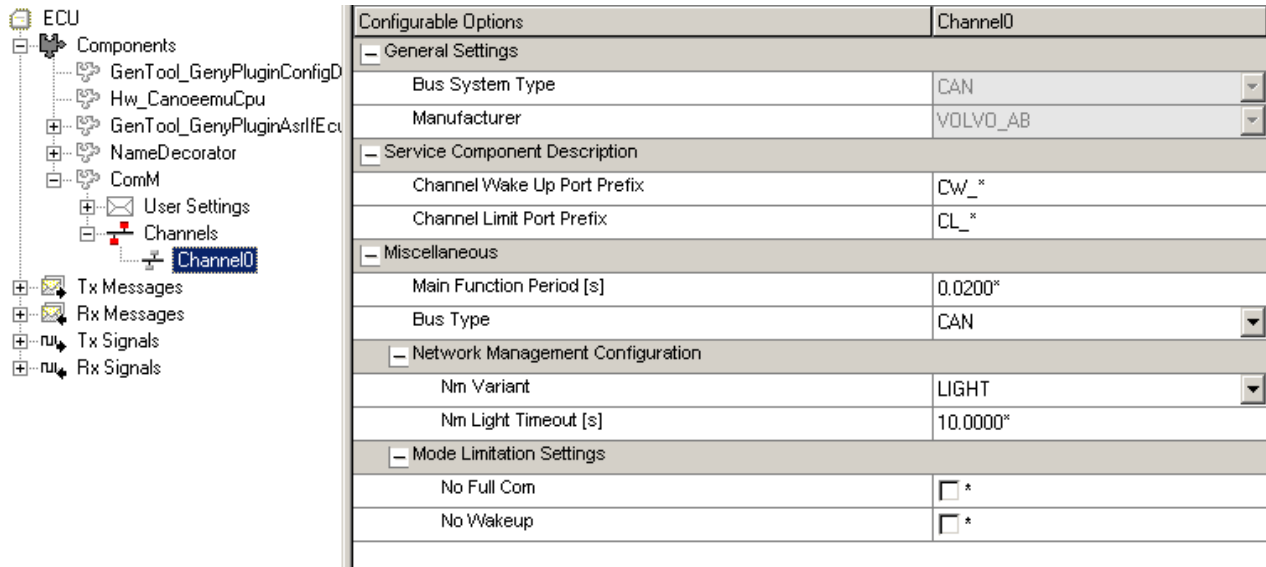This chapter describes the channel configuration of the ComM.

Figure 7-3 Channel Configuration in GENy

| Attribute Name | Configuration Variant | Value Type | Values<br>The default value is written in bold | Description |
|---|---|---|---|---|
| **Software Component Description** | | | | |
| Channel Wake Up Port Prefix | Link-Time | String | **CW_**<br>**or ANSI-C string** | Enter the prefix of the ComM_ChannelWakeUp service port. The prefix and the symbolic name of the ComM channel build the name of the service port, e.g.<br><br>Prefix = CW_<br>Channel Name = Channel0<br><br>Service Port Name =CW_Channel0 |
| Channel Limit Port Prefix | Link-Time | String | **CL_**<br>**or ANSI-C string** | Enter the prefix of the ComM_ChannelLimitation service port. The prefix and the symbolic name of the ComM channel build the name of the service port, e.g.<br><br>Prefix = CL_<br>Channel Name = Channel0<br><br>Service Port Name =CL_Channel0 |
| **Miscellaneous** | | | | |
| Main Function Period [s] | Link-time | Float | 0.0040..0.100 | Using this edit control, you are able to configure different cycle times of the Communication Manger main function ComM_MainFunction()in seconds. |
| Bus Type | | - | ■ CAN<br>■ LIN<br>■ FlexRay<br>■ ETH | Defines the Bus type |
| **Network Management Configuration** | | | | |
| Nm Variant | Link-time | - | ■ FULL<br>■ LIGHT<br>■ NONE<br>■ PASSIVE<br>■ OSEK | Defines the Nm Variant<br>■ FULL: if using the AUTOSAR NM<br>■ LIGHT: if using no AUTOSAR NM but the sleep process shall be similar to the AUTOSAR NM<br>■ NONE: No shutdown synchronization is using<br>■ PASSIVE: if using the AUTOSAR NM in passive mode<br>■ OSEK: if using only the NmOsek(direct) at the ComM channel.<br><br>**Caution**<br>If the AUTOSAR CanNm and |

| Attribute Name | Configuration Variant | Value Type | Values<br>The default value is written in bold | Description |
|---|---|---|---|---|
| | | | | NmOsek(direct) is used at the same channel then Nm variant FULL shall be selected preferred. |
| Nm Light Timeout [s] | Link-time | Float | 1.0..255.0 | Defines the Timeout after the Ready Sleep state of the ComM is left and the ComM starts the network shutdown. (Only available for the Nm variant NONE) |
| Mode Limitation Settings | | | | |
| No Full Com | Link-time | boolean | ON<br>**OFF** | Defines the initialization value of the channel mode limitation to COMM_NO_COMMUNICATION.<br><br>This value can be changed during runtime via the API ComM_LimitChannelToNoComMode(). |
| No Wakeup | Link-Time | boolean | ON<br>**OFF** | Defines the initialization value of the channel bus wake up inhibition. |

Table 7-2   Channel Configuration in GENy

### 7.1.4   User Configuration

This chapter describes the configuration of the ComM User. The user handles are used by upper layers to request ComM communication mode via the ComM service ComM_RequestComMode().

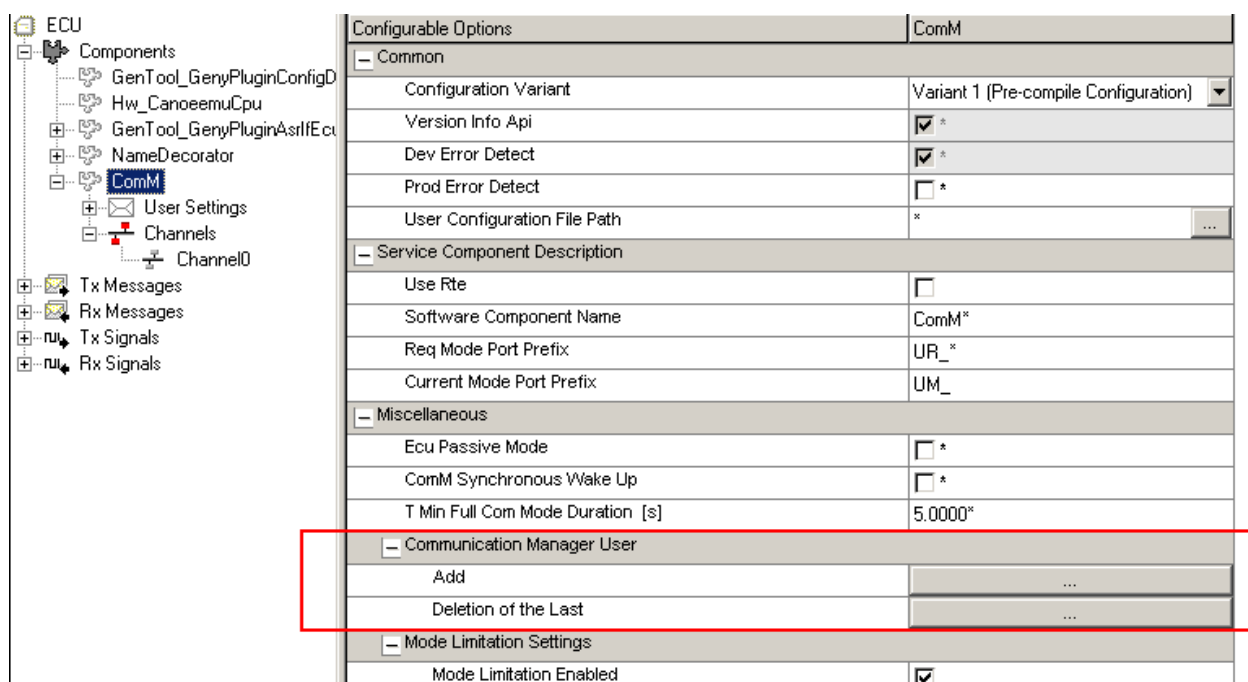| | **1st Step**<br>Add user into the configuration. This can be done inside the ComM general configuration view. |
|---|---|

Figure 7-4 ComM User configurations inside the general ComM configuration view

| | **2nd Step**<br>Assign the communication channels to the created user. This settings performs the mapping between user handle ←→corresponding communication channel(s). |



Figure 7-5 ComM User Settings

| Attribute Name | Configuration Variant | Value Type | Values<br>The default value is written in bold | Description |
|---|---|---|---|---|
| General Settings | | | | |
| Communication Manager User Handle | Link-time | String | ANSI-C string | Name of the ComM User Handle |
| ComM_RequestMode Service Port Name | - | - | - | Shows the name of the ComM_RequestMode service port name for the configured user. |
| Mode Notification | Link-time | boolean | ON<br>**OFF** | Enable/Disable the mode notification for this user. |
| Communication Channel | Link-time | boolean | **ON**<br>OFF | Assign the ComM user to the communication channel(s), which shall be handled via this user. |

Table 7-3   ComM User Settings

based on template version 3.1

# 8 AUTOSAR Standard Compliance

## 8.1 Additions/Extensions

### 8.1.1 ComM_InitMemory

This function is added to the ComM to initialize the ComM variables and set the initialization state to un-init (refer to 6.3.1).

### 8.1.2 DET Error Reporting

The ComM has advanced the following ComM DET errors:

- COMM_E_NOSUPPORTED_MODECHANGE

### 8.1.3 Nm Variant OSEK

The ComM supports additionally the Nm variant "OSEK". This Nm variant shall be used if the ComM channel uses the NmOsek(direct) instead of the AUTOSAR CanNm.

If this Nm variant is used then the ComM ensures a minimum Full communication mode time. The Full communication mode time can be configured with the configuration option "T Min Full Com Mode Duration" (7.1.2 General Configuration).

### 8.1.4 ComM Service API Return Value COMM_UNINIT

The specified ComM service API return is renamed to COMM_E_UNINIT. This must be done to prevent name clashes with the specified ComM_InitStatusType.

### 8.1.5 ComM Bus Type

The ComM supports additionally the bus type ETH for Ethernet configurations. The ETH bus type supports only the ComM Nm Variant NONE.

## 8.2 Limitation

### 8.2.1 Non-volatile data handling

The ComM uses only the NvM global block descriptor to handle the ComM non-volatile data.

# 9 Glossary and Abbreviations

## 9.1 Glossary

| Term | Description |
|---|---|
| GENy | Generation tool for CANbedded and MICROSAR components |

Table 9-1   Glossary

## 9.2 Abbreviations

| Abbreviation | Description |
|---|---|
| API | Application Programming Interface |
| AUTOSAR | Automotive Open System Architecture |
| BSW | Basis Software |
| DEM | Diagnostic Event Manager |
| DET | Development Error Tracer |
| ECU | Electronic Control Unit |
| HIS | Hersteller Initiative Software |
| MICROSAR | Microcontroller Open System Architecture (the Vector AUTOSAR solution) |
| PPort | Provide Port |
| RPort | Require Port |
| RTE | Runtime Environment |
| SRS | Software Requirement Specification |
| SWC | Software Component |
| SWS | Software Specification |
| Nm | Network Management |
| EcuM | ECU Manager |
| ComM | Communication Manager |
| CanSM | CAN State Manager |
| LinSM | LIN State Manager |
| FrSM | FlexRay State Manager |
| BusSM | Bus state manager (e.g. Lin, FlexRay and Can) |

Table 9-2   Abbreviations

# 10 Contact

Visit our website for more information on

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

**www.vector-informatik.com**