Document Title	Specification of Communication Manager
Document Owner	AUTOSAR
Document Responsibility	AUTOSAR
Document Identification No	079
Document Classification	Standard

Document Version	2.1.0
Document Status	Final
Part of Release	3.1
Revision	5

Document Change History

Date	Version	Changed by	Change Description
02.09.2010	2.1.0	AUTOSAR Administration	• Parameter name inconsistency (InhibitionStatusType <> ComM_InhibitionStatusType) • A type InhibitionStatusType is defined for the RTE interface, whilst the corresponding "module internal" type is named ComM_InhibitionStatusType. • In order to be consistent with other types like ComM_ModeType, which are named equally as "module internal" and "RTE interface" types, the RTE interface type InhibitionStatusType should be renamed to ComM_InhibitionStatusType.
23.06.2008	2.0.1	AUTOSAR Administration	Legal disclaimer revised
29.11.2007	2.0.0	AUTOSAR Administration	• Bus specific error handling (e.g. bus off handling) removed • Control of the actual bus states removed • PDU group handling removed • Initialization of Communication stack removed • Document meta information extended • Small layout adaptations made

Document Change History			
31.01.2007	1.1.0	AUTOSAR Administration	Changed features • Restart (silent com. -> full com.) now possible even if mode limitation is active • State machine changed • Sequence diagrams changed New services to upper layers • Mode indication API to RTE changed New calls to other modules • Usage of channel specific API (EcuM and ComM) to indicate that a communication channel has been woken up and has gone to sleep • API for NM control canged (Nm_PassiveStartUp, Nm_NetworkRequest, Nm_NetworkRelease) • Legal disclaimer revised • Release Notes added • “Advice for users” revised • “Revision Information” added
08.05.2006	1.0.0	AUTOSAR Administration	Initial Release

Disclaimer

Advice for users

AUTOSAR Specification Documents may contain exemplary items (exemplary reference models, "use cases", and/or references to exemplary technical solutions, devices, processes or software).

Any such exemplary items are contained in the Specification Documents for illustration purposes only, and they themselves are not part of the AUTOSAR Standard. Neither their presence in such Specification Documents, nor any later documentation of AUTOSAR conformance of products actually implementing such exemplary items, imply that intellectual property rights covering such exemplary items are licensed under the same rules as applicable to the AUTOSAR Standard.

Table of Contents

1 Introduction and functional overview

ComM35: The Communication Manager (COM Manager) is a component of the Basic Software (BSW). It is a Resource Manager which encapsulates the control of the underlying communication services.

ComM184: The COM Manager controls basic software modules (BSW) relating to communication and not software components or runable entities.

ComM54: The COM Manager collects the bus communication access requests from communication requestors (users, see chapter 2) and coordinates the bus communication access request.

ComM45: The purpose of the COM Manager is:

ComM40: 1. Simplifying the usage of the bus communication stack for the user. This includes a simplified network management handling.

ComM42: 2. Coordinating the availability of the bus communication stack (allow sending and receiving of signals) of multiple independent software components on one ECU.

Comment: A user should not have any knowledge of the hardware (e.g. on which channel he has to communicate). A user simply request a "Communication Mode" and ComM switches the communication capability of the corresponding channel on/off.

ComM43: 3. Offer an API to disable sending of signals to prevent the ECU from (actively) waking up the communication bus.

Comment: On CAN every message wakes up the bus, on FlexRay it is only possible to wake up the bus with a so called wake-up pattern.

ComM38: 4. Controlling of more than one communication bus channel of an ECU by implementing a state machine for every channel.

Comment: ComM just requests a communication mode from the corresponding Bus State Manager. The actual bus states are controlled by the corresponding Bus State Manager.

ComM174: 5. Offers the possibility to force an ECU which keeps the bus awake to "No Communication" mode (see chapter 7.2.2.2 for details).

ComM277: 6. Simplifying the resource management by allocating all resources necessary for the requested communication mode.

Comment: E.g. requesting "run mode" from ECU State Manager when a user requests "full communication mode".

2 Acronyms and definitions

ComM28:

Abbreviation / Acronym:	Description:
BSW	Basic Software
ComM	Communication Manager
DCM	Diagnostic Communication Manager
VMM	Vehicle Message Matrix
NM	Network Management
PDUR	Protocol Data Unit Router
IPDUM	IPDU Multiplexer

ComM56:

Definition:	Description:
Active wake-up	Wake-up caused by the ECU e.g. by a sensor.
Application signal scheduling	Sending of application signals according to the VMM. Scheduling of CAN application signals is performed by the communication module, scheduling of LIN application PDUs (a PDU contain signals) is performed by the LIN interface and scheduling of FlexRay application PDUs is performed by the FlexRay interface.
Bus sleep	No activity required on communication bus (e.g. CAN bus sleep).
Bus communication messages	Bus communication messages are all messages sent on the communication bus. This can be either a diagnostic message or a application message.
Diagnostic PDU scheduling	Sending of diagnostic PDUs. Scheduling of CAN diagnostic PDUs is performed by the diagnostic module, scheduling of LIN diagnostic PDUs is performed by the diagnostic module and the LIN interface and scheduling of FlexRay diagnostic PDUs is performed by the diagnostic module and the FlexRay interface.
ECU shut down	See ECU State Manager specification [6].
Independent software component	A separately developed software component performing a coherent set of functions with a minimum amount of interfaces to other software applications on an ECU. This can be e.g. a basic software component or an application software component.
COM Inhibition status	Defines whether full communication, silent communication or wake-up is allowed or not.
Passive wake-up	Wakeup by an other ECU and propagated (e.g. by bus or wakeup-line) to the ECU currently in focus.
System user	An administration functionality (a specific "user", which is generated within the internal context of the ComM) for making a default request and for overriding the user requests
User	Concept for requestors of the ECU State Manager and of the Communication Manager. A user may be a runnable entity, a SWC or even a group of SWCs which acts as a single unit towards the ECU State Manager and the Communication Manager

3 Related documentation

3.1 Input documents

ComM46:
[1] List of Basic Software Modules
AUTOSAR_BasicSoftwareModules.pdf

[2] Layered Software Architecture
AUTOSAR_LayeredSoftwareArchitecture.pdf

[3] General Requirements on Basic Software Modules
AUTOSAR_SRS_General.pdf

[4] Requirements on Mode Management
AUTOSAR_SRS_Mode_Mgmt.pdf

[5] Specification of ECU Configuration
AUTOSAR_ECU_Configuration.pdf

[6] Specification of ECU State Manager
AUTOSAR_SWS_ECU_StateManager.pdf

[7] Specification of NVRAM Manager
AUTOSAR_SWS_NVRAMManager.pdf

[8] Specification of RTE Software
AUTOSAR_SWS_RTE.pdf

[9] Specification of Network Management Interface
AUTOSAR_SWS_NMInterface.pdf

[10] Specification of Communication
AUTOSAR_SWS_COM.pdf

[11] Specification of Diagnostic Communication Manager
AUTOSAR_SWS_DCM.pdf

[12] Specification of LIN Interface
AUTOSAR_SWS_LIN_Interface.pdf

[13] Specification of FlexRay Interface
AUTOSAR_SWS_FlexRay_Interface.pdf

[14] Specification of Development Error Tracer
AUTOSAR_SWS_DET.pdf

[15] Specification of Diagnostics Event Manager
AUTOSAR_SWS_DEM.pdf

[16] Specification of CAN Transceiver Driver
AUTOSAR_SWS_CAN_TransceiverDriver.pdf

[17] Specification of CAN Interface
AUTOSAR_SWS_CAN_Interface.pdf

[18] Specification of FlexRay Transceiver Driver
AUTOSAR_SWS_FlexRayTransceiver.pdf

[19] Specification of PDU Router
AUTOSAR_SWS_PDU_Router.pdf

[20] Requirements on IPDU Multiplexer
UTOSAR_SWS_IPDUM.pdf

[21] Specification of System Services Mode Management
AUTOSAR_SystemServices_ModeManagement.pdf

[22] Specification of C Implementation Rules
AUTOSAR_SWS_C_ImplementationRules.doc

[23] Specification of LIN State Manager
AUTOSAR_SWS_LIN_StateManager.doc

[24] Specification of CAN State Manager
AUTOSAR_SWS_CAN_StateManager.doc

[25] Specification of FlexRay State Manager
AUTOSAR_SWS_FlexRay_StateManager.doc

[26] AUTOSAR Basic Software Module Description Template,
AUTOSAR_BSW_Module_Description.pdf

3.2 Related standards and norms

ComM47: Not applicable.

4 Constraints and assumptions

4.1 Limitations

ComM187: No limitations.

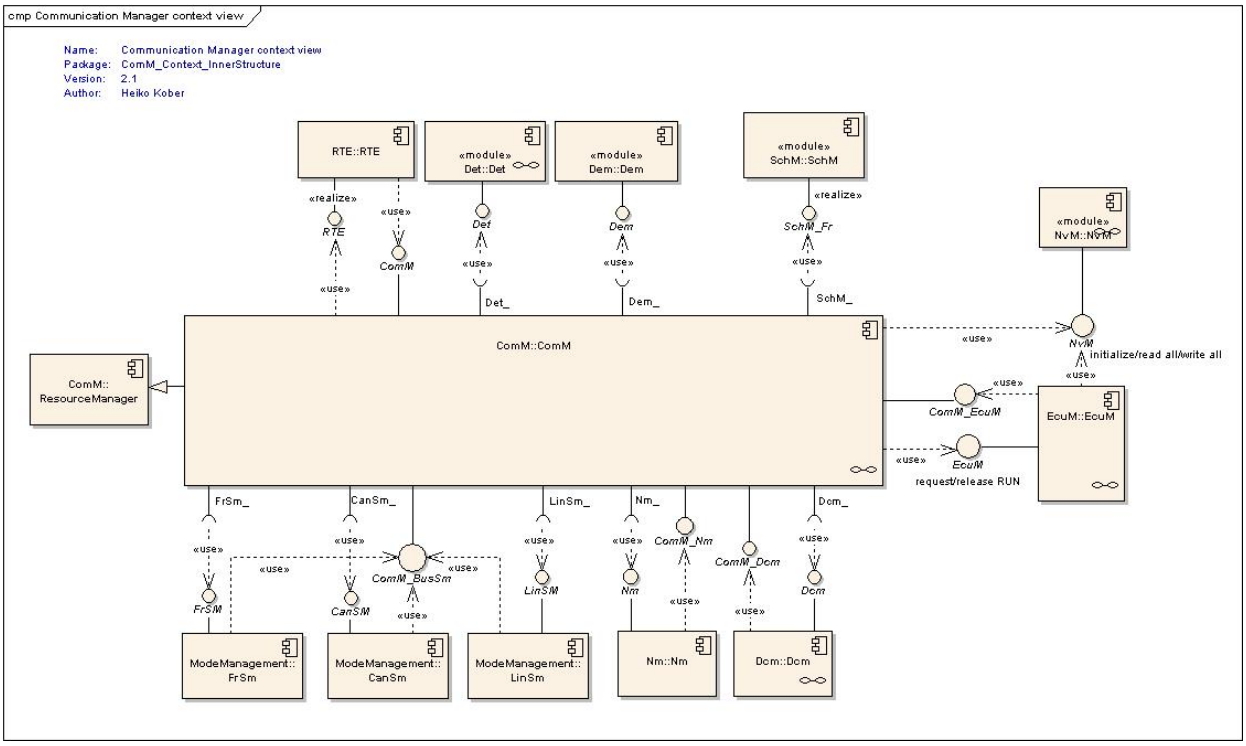
4.2 Applicability to car domains

ComM186: No restrictions.

5 Dependencies to other modules

ComM44: A context view which shows the Communication Manager and the dependencies to other modules is shown in figure 5.1.

ComM36:



ComM37: Figure 5.1: Communication Manager context view

ComM50: The ComM requests the communication capabilities, requested from the users, from the Bus State Managers.

5.1 File structure

5.1.1 Code file structure

ComM503: The code file structure shall not be defined within this specification completely. At this point it shall be pointed out that the code-file structure shall include the following files named:

- ComM_Lcfg.c - for link time configurable parameters and
- ComM_PBcfg.c - for post build time configurable parameters.

These files shall contain all link time and post-build time configurable parameters.

ComM504: The code file structure shall not be defined within this specification.

5.1.2 Header file structure

ComM466: ComM shall use the Standard header files (For details refer to AUTOSAR General Requirements on Basic Software Modules [3]). It is not allowed to redefine AUTOSAR integer data types.

ComM507: ComM shall include the Dem.h file. By this inclusion the APIs to report errors as well as the required Event Id symbols are included. This specification defines the name of the Event Id symbols which are provided by XML to the DEM configuration tool. The DEM configuration tool assigns ECU dependent values to the Event Id symbols and publishes the symbols in Dem_IntErrId.h.

ComM506: ComM shall include the header files of the modules providing interfaces to ComM (see figure 5.1).

ComM463: ComM shall provide in addition to ComM_Lcfg.c and ComM_PBcfg.c at least the following files:

1.ComM header file:	ComM.h
2.ComM callback declarations	ComM_Nm.h, ComM_EcuM.h, ComM_Dcm.h, ComM_BusSm.h
3.ComM source file:	ComM.c
4.ComM configuration file:	ComM_Cfg.h
5.ComM Schedule Manager file	SchM_ComM.h

Rationale: Source code and configuration are strictly separated. User defined configurations will not imply the change of the original source code.

5.2 AUTOSAR Runtime Environment (RTE)

ComM343: Every user can request a so called "communication mode". The RTE propagates the user request to the ComM and the mode indications from the ComM to the users (for details refer to [8]).

5.3 ECU State Manager (EcuM)

ComM344: The ComM requests the required resources from the EcuM by requesting "run mode" to avoid that the EcuM shuts down the ECU. This has the advantage that users which require communication do not have to request "RUN" separately

Comment: EcuM shutdown the ECU if no user requests run.

5.4 NVRAM Manager

ComM188: The ComM shall use NVRAM Manager to store non-volatile data. For details on initial values of the NVRAM data refer to chapter 10.

5.5 Diagnostic Communication Manager (DCM)

ComM346: DCM performs the scheduling of diagnostic PDUs. DCM acts as a user by requesting "Full Communication" if diagnostics shall be performed. DCM

does not provide an API to start/stop sending and receiving but guarantees that the communication capabilities are according to the ComM communication modes.

5.6 LIN State Manager

ComM348: LIN State Manager controls the actual states of the LIN bus which corresponds to a communication mode of ComM. ComM requests a communication mode from the LIN State Manager and the LIN State Manager shall map the communication mode to a bus state.

5.7 CAN State Manager

ComM353: CAN State Manager controls the actual states of the CAN bus which corresponds to a communication mode of ComM. ComM requests a communication mode from the CAN State Manager and the CAN State Manager shall map the communication mode to a bus state.

5.8 FlexRay State Manager

ComM349: FlexRay State Manager controls the actual states of the FlexRay bus which corresponds to a communication mode of ComM. ComM requests a communication mode from the FlexRay State Manager and the FlexRay State Manager shall map the communication mode to a bus state.

5.9 Network Management (NM)

ComM347: ComM shall use the NM to synchronize the control of communication capabilities across the network (synchronous start-up and shutdown).

5.10 Diagnostic Event Manager (Dem) and Development Error Tracer (Det)

ComM580: Dem provides services to indicate production errors and Det provides services to store development errors (see chapter 7.8)

6 Requirements traceability

ComM228: Document: AUTOSAR requirements on Basic Software, general
ComM230:

Requirement	*Description*	*Satisfied by*
BSW003	Version identification	ComM280
BSW004	Version check	ComM418
BSW006	Platform independency	ComM462
BSW007	HIS MISRA C	ComM462
BSW101	Initialization interface	ComM146
BSW158	Separation of configuration from implementation	ComM464
BSW159	Tool-based configuration	ComM457
BSW160	Human-readable configuration data	ComM460
BSW164	Implementation of interrupt service routines	ComM458
BSW167	Static configuration checking	ComM419
BSW171	Configurability of optional functionality	ComM555 ComM558 ComM559 ComM561
BSW00300	Module naming convention	ComM462
BSW00301	Limit imported information	ComM462
BSW00302	Limit exported information	ComM462
BSW00304	AUTOSAR integer data types	ComM466
BSW00305	Self-defined data types naming convention	ComM462
BSW00306	Avoid direct use of compiler and platform specific keywords	ComM462
BSW00307	Global variables naming convention	ComM462
BSW00308	Definition of global data	ComM462
BSW00309	Global data with read-only constraint	ComM462
BSW00310	API naming convention	ComM462
BSW00312	Shared code shall be reentrant	ComM462
BSW00318	Format of module version numbers	ComM280
BSW00321	Enumeration of module version numbers	ComM469
BSW00323	API parameter checking	ComM234
BSW00327	Error values naming convention	ComM234
BSW00328	Avoid duplication of code	ComM462
BSW00329	Avoidance of generic interfaces	ComM462
BSW00330	Usage of macros / inline functions instead of functions	ComM462
BSW00331	Separation of error and status values	ComM649 ComM494
BSW00334	Provision of XML file	ComM460
BSW00335	Status values naming convention	ComM494
BSW00336	Shutdown interface	ComM147
BSW00337	Classification of errors	ComM234
BSW00338	Reporting of development errors	ComM270
BSW00339	Reporting of production relevant error status	ComM271

Requirement	Description	Satisfied by
BSW00342	Usage of source code and object code	ComM459
BSW00345	Pre-compile-time configuration	ComM456
BSW00346	Basic set of module files	ComM463
BSW00347	Naming separation of different instances of BSW drivers	ComM462
BSW00350	Development error detection keyword	ComM555
BSW00355	Do not redefine AUTOSAR integer data types	ComM466
BSW00358	Return type of init() functions	ComM146
BSW00359	Return type of callback functions	ComM468
BSW00360	Parameters of callback functions	ComM468
BSW00369	Do not return development error codes via API	ComM649
BSW00370	Separation of callback interface from API	ComM36
BSW00371	Do not pass function pointers via API	ComM462
BSW00373	Main processing function naming convention	ComM429
BSW00374	Module vendor identification	ComM280
BSW00376	Return type and parameters of main processing functions	ComM429
BSW00377	Module specific API return types	ComM649
BSW00379	Module identification	ComM280
BSW00380	Separate C-Files for configuration parameters	ComM503
BSW00383	List dependencies of configuration files	ComM36
BSW00384	List dependencies to other modules	ComM36
BSW00385	List possible error notifications	ComM234
BSW0386	Configuration for detecting an error	ComM381 ComM380 ComM377 ComM234
BSW00387	Specify the configuration class of callback function	ComM620
BSW00435	Module Header File Structure for the Basic Software Scheduler	ComM463, Com M36
BSW00388	Introduce containers	ComM585
BSW00389	Containers shall have names	ComM585
BSW00390	Parameter content shall be unique within the module	ComM585
BSW00391	Parameter shall have unique names	ComM585
BSW00392	Parameters shall have a type	ComM585
BSW00393	Parameters shall have a range	ComM585
BSW00394	Specify the scope of the parameters	ComM585
BSW00395	List the required parameters (per parameter)	ComM562 ComM565
BSW00396	Configuration classes	ComM585
BSW00401	Documentation of multiple instances of configuration parameters	ComM585
BSW00402	Published information	ComM280
BSW00406	Check module initialization	ComM242
BSW00407	Function to read out published parameters	ComM370
BSW00408	Configuration parameter naming convention	ComM585
BSW00409	Header files for production code error IDs	ComM36

Requirement	Description	Satisfied by
BSW00410	Compiler switches shall have defined values	ComM585
BSW00411	Get version info keyword	ComM622
BSW00412	Separate H-File for configuration parameters	ComM463
BSW00414	Parameter of init function	ComM146
BSW00415	User dependent include files	ComM463
BSW00416	Sequence of Initialization	ComM276
BSW00419	Separate C-Files for pre-compile time configuration parameters	ComM503

ComM229: Document: AUTOSAR requirements on Basic Software, cluster Communication Manager

ComM231:

Requirement	Description	Satisfied by
BSW049	Initiating wake-up and keeping awake physical channels	ComM129
BSW09071	Limit Communication Manager modes	ComM357 ComM303
BSW09078	Coordinating communication requests	ComM283
BSW09079	Transparent relationship between software components and physical channels	ComM191 ComM159
BSW09080	Physical channel independency	ComM51
BSW09081	API for requesting communication	ComM110
BSW09083	Support of different communication modes	ComM190
BSW09084	API for querying the current communication mode	ComM79 ComM83
BSW09085	Indication of communication mode changes	ComM172 ComM91
BSW09087	Proxy communication request after wake-up	ComM207
BSW09088	Handling of different physical channel types	ComM281 ComM322
BSW09089	Preventing waking up physical channels	ComM302
BSW09090	User-to-channel relationship	ComM159
BSW09133	Assigning physical channels to the Communication Manager	ComM327
BSW09141	Configuration of physical channel wake-up prevention	ComM559
BSW09149	API for querying the requested communication mode	ComM79
BSW09155	Counting of inhibited communication requests	ComM138
BSW09156	API to retrieve the number of inhibited "Full Communication" mode requests	ComM224 ComM108
BSW09157	Revoke Communication Manager mode limitation	ComM156, ComM163, ComM124
BSW9172	Evaluation of current communication mode	ComM176

7 Functional specification

ComM283: The ComM shall simplify the resource management for the users. The user simply requests a communication mode and ComM shall request all the required resources.

ComM686: ComM shall coordinate multiple independent user requests according to the "highest wins" strategy. This means that if at least one user requests a higher communication mode then this is the target communication mode.

ComM484: The highest communication mode shall be "Full Communication" and the lowest communication mode shall be "No Communication". "Silent Communication" shall be in-between.

ComM281: The communication manager shall be able to handle different bus types and internal communication.

Comment: Currently only CAN, LIN and FlexRay is covered by configuration.

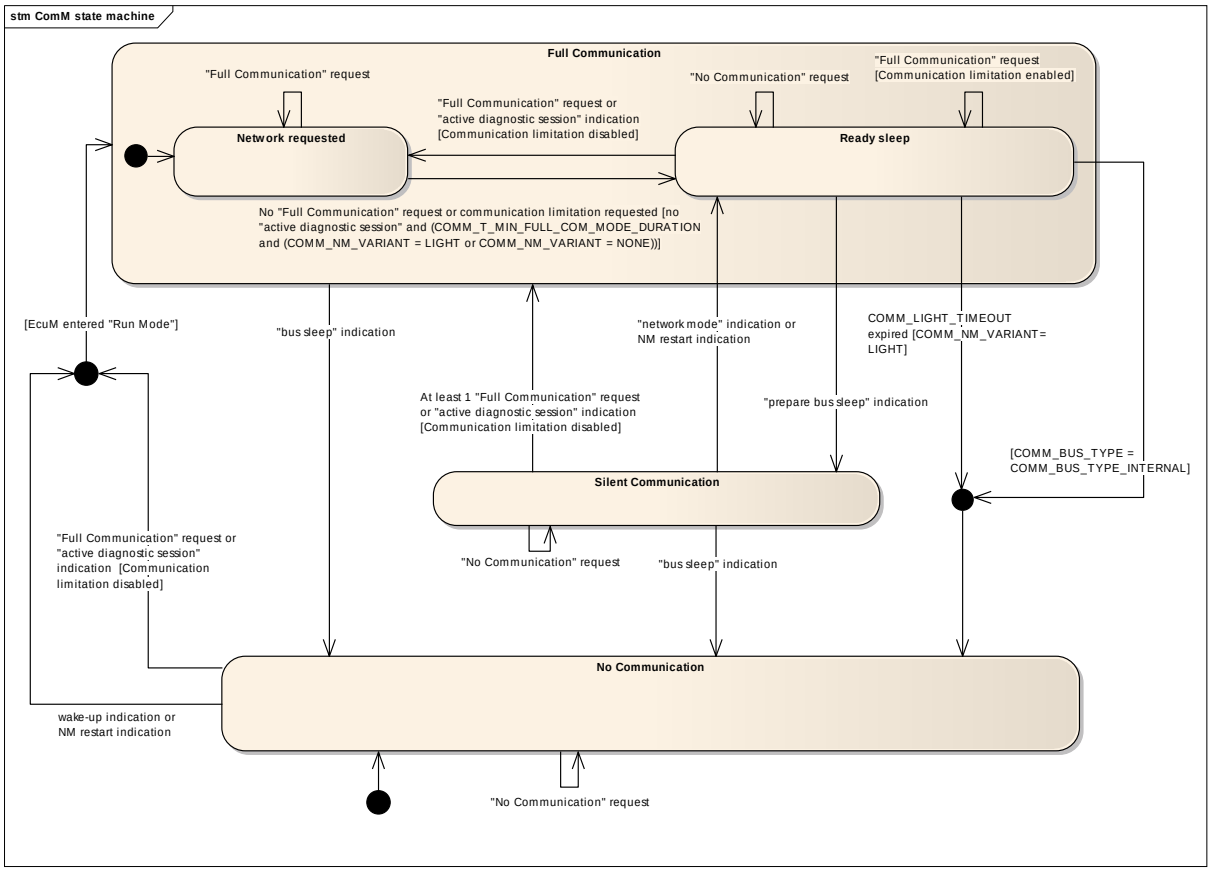
7.1 State machine

ComM51: The COM Manager shall implement the state machine shown in figure 7.1 for every communication channel independently.

Rationale: Needed communication capability of channels may be different, thus the controlling must be independent.

Use Case: On an ECU with CAN and LIN channel only LIN requires full communication to request e.g. sensor values.

ComM52:



ComM53: Figure 7.1: COM Manager state machine

ComM786: Every transition and the behavior in the states is explained in the chapters resp. requirements listed in Table 7.1.

ComM787:

State resp. transition	Requirement resp. chapter
No Communication	7.1.1
Silent Communication	7.1.2
Full Communication	7.1.3
No Communication → Full Communication	ComM207, ComM694, ComM784, ComM128
Network Requested → Ready Sleep	ComM665
Ready Sleep → Network Requested	ComM479
Ready Sleep → Silent Communication	ComM299
Ready Sleep → No Communication	ComM610, ComM671
Full Communication → No Communication	ComM637
Silent Communication → Full Communication	ComM785
Silent Communication → Ready Sleep	ComM296
Silent Communication → No Communication	ComM295

ComM788: Table 7.1: Link to detailed explanation of the state machine resp. transition

ComM190: The state machine shall consist of the 3 main states "No Communication", "Silent Communication" and "Full Communication" and the sub-states "Network requested" and "Ready Sleep".

Rationale: "Ready Sleep" and "Silent Communication" is necessary to synchronize a communication shutdown on the bus. If only one ECU switches the communication off, the others store errors because this ECU stops sending application signals.

Comment: The main states present an abstracted status of communication capabilities per channel, which are in focus of the users interests. The sub-states represent intermediate states which perform activities to support a synchronized transition with external partners and managing protocols (e.g. NM).

ComM485: The default state of the state machine shall be "No Communication".

ComM151: It shall only be possible to request the main states "No Communication" and "Full Communication" (*ComM_RequestComMode*).

Rationale: Only the main states are related to the modes that can be requested by a user. Sub-states are only necessary for synchronization with AUTOSAR NM.

Comment: The majority don't see the necessity to have the possibility to request "Silent Communication" thus it is removed since release 2.0.

ComM191: The functionality of the state machine shall be encapsulated from the users which means that the user needs no information about the mechanisms and rules ("highest wins" strategy) of the state machine.

ComM471: Each user shall request exactly one state at any time.

ComM500: User requests shall not be queued. The latest user request of the same user shall overwrite an old user request even if the request is not finished.

ComM289: An "active diagnostic" indication (*ComM_DCM_ActiveDiagnostic*) from the DCM shall be treated as an "Full Communication" request for all channels.

Rationale: To avoid that the bus falls asleep during a diagnostic session.

ComM80: ComM shall allow querying the communication mode requested by a particular user.

Use Case: User 1 request "Full Com." -> user 2 request "Full Com." -> User 1 release "Full Com." -> "Full Com. shall still be active.

ComM84: It shall be possible to query the actual communication mode of a channel from the ComM. ComM shall propagate this request to the corresponding Bus State Manager.

Rationale: State request have to be propagated to the corresponding Bus State Manager since ComM does not control the actual bus state.

Comment: This feature is not used by a "normal SWC" because they don't have knowledge about channels. This feature is necessary for privileged SWC which (have to) know about the system topology e.g. system diagnostic functions.

ComM92: There shall be one target state (evaluated according ComM283) per channel. This target state can differ temporarily from the actual state controlled by the corresponding Bus State Manager.

Comment: Mode switching by the corresponding Bus State Manager takes time and a mode inhibition can be active.

ComM472: Main state changes (see ComM190) shall be indicated to the users with the corresponding notifications (see chapter 8.6.1.10).

Comment: If more than one user is related to the corresponding state machine, ComM has to perform a "fan out".

ComM210: An overview of the requested communication capabilities in the corresponding mode is shown in table 7.2.

ComM211:

Communication Mode	Message Transmission	Message Reception	NM (COMM_NM_VARIANT=FULL)	Wake-up/Restart capability
Full Communication				
Network Requested	On	On	Bus communication requested	N/A
Ready Sleep	On	On	Bus communication released	N/A
Silent Communication	Off	On	Bus communication released	• User/diagnostic request • Network indication
No Communication	Off	Off	Bus communication released	• User/diagnostic request • Passive wake-up

ComM232: Table 7.2: Granted communication capabilities in the corresponding modes

7.1.1 Behavior in state "No Communication"

ComM313: ComM shall not indicate the “No Communication” mode to the RTE if ComM enters “No Communication” mode by default after the initialization.

Rationale: RTE is not yet initialized at this point.

Comment: A state change (e.g. No Com. --> Full Com.) is triggered by a user which is usually a SWC. A SWC is only active after RTE is initialized thus there
is no requestor before RTE is started thus no state change can occur except entering the state machine per default.

ComM70: In "No Communication" mode, no bus communication for the corresponding channels shall be requested.
Use Case: ECU is performing control functions locally without participation in bus communication.
Comment: The mode is local for one channel thus the ECU may communicate via other channels.
ComM73: Transmission and reception capability of the corresponding channels shall be switched off. This shall be performed by requesting the corresponding communication mode from the corresponding Bus State Manager (*XXSM_RequestComMode(network, mode)*).
Rationale: The "No Communication" mode forbids sending and receiving of PDUs for the corresponding channels.
ComM784: The state machine shall switch to state "Full Communication" if at least one user requests "Full Communication" or DCM indicates "active diagnostic session" and communication limitation (see chapter 7.2.2) is disabled.
ComM688: ComM shall request "No Communication" from the corresponding State Manager even if a user requests "Full Communication" or DCM indicates "active diagnostic" (*ComM_DCM_ActiveDiagnostic*) if the previous state was "Full Communication". The state machine shall switch to state "Full Communication" if at least one user requests "Full Communication" or DCM indicates "active diagnostic session" and communication limitation (see chapter 7.2.2) is disabled after executing the "No Communication" request.
Rationale: FlexRay shutdown can not be interrupted to avoid partial networks.
ComM288: ComM shall request *Nm_NetworkRelease()* from Network Management for the corresponding channel if ComMNmVariant = FULL is configured.
ComM130: The COM Manager shall release the "Run Mode" request for the corresponding state machine (resp. channel) from ECU State Manager after entering the No Communication state.
Rationale: No Communication mode requires no ECU resources from ECU State Manager.
ComM207: ComM shall switch the corresponding state machine (resp. channel) to state "Full Communication" if a passive wakeup-indication of a channel is indicated by EcuM or NM indicates a restart (ComM_Nm_RestartIndication).
Rationale: It must be guaranteed that communication starts as soon as possible after a bus wake up.
Comment: The state machine switches immediately to sub-state "Network requested" after entering "Full Communication" mode if no user requests "Full Communication" mode. AUTOSAR NM resp. ComM state duration extension functionality extensions guarantees that this state is hold for a configurable time to prevent toggling and to overcome the init-/start-up time of the system, before possible user requests occur.
ComM694: ComM shall switch all state machines (resp. channels) to state "Full Communication" if a passive wake up indication of a channel is indicated by EcuM and COMM_SYNCHRONOUS_WAKE_UP is enabled.
ComM128: The COM Manager shall request "Run Mode" from the ECU State Manager for the corresponding state machine (resp. channel) before entering "Full Communication" if "Run Mode" is not already entered. The

state machine shall stay in "No Communication" mode until the ECU state Manager indicates "Run Mode" for the corresponding channel.

Rationale: "Full Communication mode" requires "Run Mode" from the ECU state Manager.

7.1.2 Behavior in state "Silent Communication"

ComM785: The state machine shall switch to state "Full Communication" if at least one user requests "Full Communication" or DCM indicates "active diagnostic session" and communication limitation (see chapter 7.2.2) is disabled.

ComM71: Reception capability of the corresponding channels shall be switched on. This shall be performed by requesting the corresponding communication mode from the corresponding Bus State Manager.

Rationale: The "Silent Communication" mode permits receiving of PDUs for the corresponding channels.

Comment: It may happen that nothing is received (e.g. during bus off) despite receiving capability is switched on.

ComM72: Transmission capability of the corresponding channels shall be switched off. This shall be performed by requesting the corresponding communication mode from the corresponding Bus State Manager.

Rationale: The "Silent Communication" mode forbids sending of PDUs for the corresponding channels.

Use Case: shut down coordination with means of NM (prepare bus sleep state).

ComM295: The state machine shall switch to "No Communication" state after "bus sleep mode" indication from the Network Management.

ComM296: The state machine shall switch to "Ready Sleep" state after "network mode" or "restart" indication from the Network Management.

7.1.3 Behavior in state "Full Communication"

ComM69: Transmission and reception capability of the corresponding channels shall be switched on. This shall be performed by requesting the corresponding communication mode from the corresponding Bus State Manager.

Rationale: The "Full Communication mode" permits sending and receiving of bus communication PDUs for the corresponding channel.

ComM637: The state machine shall switch to "No Communication state" after "bus sleep mode" indication from the Network Management.

Rationale: A user may request to keep the bus awake "too late" (NM is not able to send a vote to keep the bus awake because the cluster already agreed to shutdown).

7.1.3.1 Network Requested state

ComM129: The ComM shall request *Nm_NetworkRequest()* from the Network Management for the corresponding NM channels if a user requests "Full

Communication" or the DCM indicates "active diagnostic" and ComMNmVariant = FULL is configured.

ComM665: The ComM shall request *Nm_PassiveStartup()* from the Network Management for the corresponding NM channels if EcuM (see [6]) indicated a passive wakeup.

ComM478: The state machine shall immediately switch to "Ready Sleep" state if no user requests "Full Communication" or communication limitation (see chapter 7.2.2) is requested and DCM does not indicate "active diagnostic" (*ComM_DCM_ActiveDiagnostic*).

ComM68: The "Full Communication" mode is the mode with the highest ECU activity.

7.1.3.2 Ready Sleep state

ComM133: The ComM shall request *Nm_NetworkRelease()* from the Network Management for the corresponding NM channels if ComMNmVariant = FULL is configured.

ComM299: The state machine shall immediately switch to "Silent Communication" after "prepare bus sleep mode" indication from the Network Management if ComMNmVariant = FULL or ComMNmVariant = PASSIVE is configured.

ComM610: The state machine shall immediately switch to "No Communication" after COMM_LIGHT_TIMEOUT is expired and ComMNmVariant = LIGHT is configured.

ComM671: The state machine shall immediately switch to "No Communication" if COMM_BUS_TYPE = INTERNAL is configured.

ComM479: The state machine shall immediately switch to "Network Requested" state if a user requests "Full Communication" or the DCM indicates "active diagnostic" and communication limitation (see chapter 7.2.2) is disabled.

7.2 Extended functionality

ComM470: Extended functionality except "state duration extension" shall be configurable per feature (for details see chapter 10).

Rationale: Some software components must be independent from limitations.

Use Case: Door module must not be limited to "No Communication" mode to avoid that the driver can not open the door because of a mode limitation because of a low battery.

Comment: Configurable with ComM_ECU_Group_Classification (see 10.1.2).

7.2.1 State duration extensions

ComM205: It shall be ensured that the "Network Requested" state for each channel is left not earlier than the time duration COMM_T_MIN_FULL_COM_MODE_DURATION after entering the Full Communication state for the corresponding channels.

Rationale: The ECU must be at least for COMM_T_MINIMUM_FULL_COM_MODE_DURATION in Full communication mode to prevent toggling.

ComM311: The state duration extensions shall only be active if the corresponding channel does not use an AUTOSAR NM (ComMNmVariant = LIGHT or ComMNmVariant = NONE).
Rationale: Avoiding of redundant functionality because AUTOSAR NM also ensures this functionality.

7.2.2 Communication inhibition

ComM300: The purpose of mode inhibition is to limit the communication capabilities. For details see chapter 7.2.2.1 and 7.2.2.2.
ComM301: ComM shall offer interfaces to request and release the corresponding mode inhibitions.
Comment: ComM doesn't care about who requests the mode inhibition but it is not a "normal" SWC. It is a privileged SWC or a OEM specific BSW.
ComM488: It shall be possible to enable and disable the mode inhibition for each channel (state machine) independently. This functionality shall not be used by ComM itself.
ComM361: ComM shall not support multiple (re-entrant) "Limit to No Communication" requests.
Comment: The system designer must ensure that there is only one "master" which controls the mode inhibitions to avoid conflicting requests.
ComM304: The status of the user requests shall be stored. The status shall also be updated if a user releases a request during an active mode inhibition.
Rationale: User requests shall be granted if the inhibition gets disabled.
Comment: Amount of active user requests from different users.
ComM182: The communication inhibition shall get temporarily inactive during an active diagnostic session.
Rationale: ECUs must not fall asleep during an active diagnostic session.
Comment: DCM indicates the start of a active diagnostic session with *ComM_ActiveDiagnostic()* and the end of a diagnostic session with *ComM_InactiveDiagnostic()* .

7.2.2.1 Bus wake Up inhibition

ComM329: Bus wake Up Inhibition in context of the Communication Manager means, that the Communication Manager should take precautions against awaking other ECU's by starting the communication.
Rationale: Awaking other ECU's by communication should be avoided because it is assumed that the ECU wakes up the bus because of an error (e.g. broken sensor).
Use Case: An error was detected on signal path of an active wakeup line and this non reliable wakeup-source should not be able to awake the whole system any more. A SW-Component, that controls error-reactions could set the wakeup inhibition-status of related COM-channels, that usually get communication-requests from SW-Components as the consequence of this event. This corrupts the forwarding of communication system-wide, based on unreliable wakeup events.

Or in case of application specific system control, there is a SW-Component that should switch off forwarding system wide wakeup's by communication under conditions like i.e. transport mode.

ComM302: Bus wake up Inhibition shall be performed by ignoring user requests.

Comment: Ignoring user requests mean accepting the requests but not execute them due to mode inhibtion. The highest win strategy would apply immediately as soon as mode inhibition is switched off.

ComM218: A communication request (Full communication) by a user shall be inhibited if the COM Inhibition status is equal to ComMNoWakeup for the corresponding channel and the current state of the channel is "No Communication" or "Network released".

Rationale: The inhibition should not get active, if the inhibition-status is set but the COM-channel is already active.

ComM219: The inhibition shall not get active if the current communication state is "Full Communication" or "Silent Communication".

Rationale: The bus is already awake if the current communication state is Full Communication. The Bus is not yet switched of in "Full Communication" state.

ComM66: ComM shall never inhibit the passive wake-up capability.

Rationale: It must be always possible to react on bus wake-ups indicated by the ECU State Manager.

Comment: Reception is switched off in "No Communication" mode but the wake-up capability is switched on.

ComM157: Inhibition must be stored non volatile.

Rationale: Information must be available during startup, before the communication is active (Full Commiunication mode entered). Changing or query is only possible after startup up with active communication (usually the "master" who decides that the inhibition is active or not is not on the same ECU).

ComM625: The status of the user requests shall also be updated if a user releases a request.

7.2.2.2 Limit to No Communication mode

ComM303: Limit to No Communication mode is performed by switching to "Ready Sleep" state to initiate a shutdown despite user requests for "Full Communication" mode and ignoring new "Full Communication" mode requests.

Rationale: Forcing into No Communication mode is needed to shut down software components which keeps the bus awake.

ComM424: Limit to "No Communication" mode shall only be performed if the current state is "Network Requested". Requests in other states shall be ignored.

ComM215: All active user requests for channel X shall be ignored if the COM Inhibition status is equal to ComMNoCom for the corresponding channel to guarantee entering the "No Communication" state for channel X.

ComM216: All active user requests for all channels shall be ignored if the COM Inhibition status is equal to ComMNoCom to guarantee entering the "No Communication" state for all channels.

ComM355: ComM shall force an ECU reset by selecting the shutdown target "reset" (*EcuM_SelectShutdownTarget(ECUM_STATE_RESET)*) and calling

EcuM_KillAllRUNRequests after entering "No Communication" mode if configured (COMM_RESET_AFTER_FORCING_NO_COMM).

Rationale: It is assumed that a faulty user will not release his "Full Communication" request without a re-initialization. Keeping the "Full Communication" request active leads to a toggling between network shutdown and network startup.

Use Case: It is assumed that a faulty ECU keeps the bus awake. As a consequence a "network master" decides to force all ECUs to go to sleep.

ComM582: ComM shall clear the user requests after all the channels that belong to the corresponding user entering "No Communication" mode.

Rationale: Stored (faulty) user requests which are assumed to keep the bus awake must be cleared.

ComM105: The COM Inhibition status ComMNoCom for the corresponding channel and ComMNoCom (limit all channels to No Communication) shall not be stored persistent.

Rationale: If this limitation is stored persistent, the SWC can never switch to Silent or Full communication, because it can never receive signals, thus the inhibition can never be switched off.

7.3 Bus communication management

ComM402: ComM shall use the corresponding interfaces of the Bus State Managers to control the communication capabilities.

ComM664: ComM shall omit calls to control the communication capabilities if COMM_BUS_TYPE = COMM_BUS_TYPE_INTERNAL.

Rationale: Internal communication has no corresponding bus interface.

7.4 Network management dependencies

ComM599: ComM shall support the shutdown synchronization variants (configured with ComMNmVariant, see chapter 10.2.6) LIGHT and FULL described in table 7.3.

Comment: Only variant FULL and PASSIVE guarantees a synchronized shutdown between all nodes of a network.

ComM600:

NM variant	Wake-up capability	Shutdown synchronization
NONE	Wake-up by bus or wake-up line possible	No shutdown synchronization by ComM. Shutdown by switching off the power of the ECU.
LIGHT	Wake-up by wake-up line required	Shutdown synchronization by ComM with means of a timeout (configured with COMM_NM_LIGHT_TIMEOUT, see chapter 10.2.2)
PASSIVE	Wake-up by bus or wake-up line possible. ECU is not allowed to keep the bus awake	Shutdown synchronization by ComM with means of AUTOSAR NM.
FULL	Wake-up by bus or wake-up line possible. ECU is allowed to keep the bus awake.	Shutdown synchronization by ComM with means of AUTOSAR NM.

Comment: A synchronized shutdown is not possible with the LIGHT variant thus the ECU may continuously restart ("toggle") because of a message from a node shutting down later.

ComM601: Table 7.3: Network management variants supported by ComM
ComM602: ComM shall omit calls of NM services if ComMNmVariant = LIGHT or ComMNmVariant = NONE is configured.
Rationale: NM services are not available if no NM is available.
ComM667: ComM shall omit to call *Nm_NetworkRequest()* from NM if ComMNmVariant = PASSIVE is configured.
Rationale: Service *Nm_NetworkRequest()* is not available.

7.5 Bus error management

7.5.1 Network Start Indication

ComM377: ComM shall set the error COMM_E_NET_START_IND_CHANNEL_<X> if NM indicates ComM_Nm_NetworkStartIndication for channel X.
ComM583: ComM shall switch channel X to "Full Communication" if NM indicates ComM_Cbk_Nm_NetworkStartIndication for channel X.
Use Case: A node send a NM message in "Prepare Bus Sleep" state but other nodes are already in "Bus Sleep" state because of "race conditions"

7.6 Test support requirements

7.6.1 Inhibited Full Communication Request Counter

ComM138: The Com Manager shall provide one counter for all rejected "Full Communication" mode requests. It counts user requests which cannot be fulfilled because the system has inhibited communication modes.

Rationale: The counter is used for detecting latent software problems relating to unmotivated communication bus wakeups
ComM140: The counter shall be stored in non-volatile memory.
ComM141: The range of the counter shall be 0 to 65535.
ComM142: The counter shall stop incrementation if the maximum counter value is reached.
ComM143: It shall be possible to read out and reset the counter value by a Com Manager API call.
Use Case: It shall be possible to read out and reset the current status of the counter by a diagnostic service.

7.7 Error classification

ComM508: Values for production code Event Ids are assigned externally by the configuration of the Dem. They are published in the file Dem_IntErrId.h and included via Dem.h.
ComM509: Development error values are of type uint8.
ComM269: The errors and exceptions of table 7.4 shall be detectable by the ComM depending on its build version (development/production mode).

ComM234:

Type or error	***Relevance***	***Related error code***	***Value [hex]***
API service used without module initialization	Development	COMM_E_NOT_INITED	0x1
API service used with wrong parameters (e.g. a NULL pointer)	Development	COMM_E_WRONG_PARAMETERS	0x2
Provided API services of other modules returned with an error.	Development	COMM_E_ERROR_IN_PROV_SERVICE	0x3
Reception of NM messages in “No Communication” mode	Production	COMM_E_NET_START_IND_CHANNEL_<X> with X being the number of the channel.	Assigned by DEM

ComM397: Table 7.4: Error classification
ComM621: None of the production errors of table 7.2 shall be healable.
ComM612: If not initialized, ComM shall reject every API service apart from ComM_Init; the called function shall not be executed, but instead of that it shall report COMM_E_NOT_INITED to the Development Error Tracer and the calling function.
ComM328: The ComM files shall check the consistency between the header, C and configuration files by means of the published parameters (see chapter 10.2) during compilation according to BSW004. This is to guarantee the consistency of the files and the code generator to the same release.

7.8 Error detection

ComM511: The detection of development errors is configurable (*ON* / *OFF*) at pre-compile time. The switch <*MODULE PREFIX*>*_DEV_ERROR_DETECT* (see chapter 10) shall activate or deactivate the detection of all development errors.

ComM512: If the *ComMDevErrorDetect* switch is enabled API parameter checking is enabled. The detailed description of the detected errors can be found in chapter 7.7 and chapter 8.

ComM513: The detection of production code errors cannot be switched off.

7.9 Error notification

ComM270: Detected development errors will be reported to the error hook of the Development Error Tracer (DET, see [14]) if the pre-processor switch *ComMDevErrorDetect* is set (see chapter 10).

ComM271: Production errors shall be reported to the Diagnostics Event Manager [15].

ComM515: Only the error case (‘failed’) shall be reported to Dem.

7.10 Non functional requirements

ComM458: ComM is not allowed to implement interrupt service routines.

Rationale: The implementation of interrupt service routines is highly microcontroller dependent.

ComM640: ComM is not allowed to use operating system timers and resources directly.

ComM459: It shall be possible to integrate ComM delivered as source or object code into the AUTOSAR stack.

Rationale: Allow IP protection and guaranteed test coverage : object code an allow high efficiency and configurability at system generation time (by integrator) : source code.

ComM462: ComM shall be implemented according the AUTOSAR Software Module Design Requirements (For details refer to AUTOSAR General Requirements on Basic Software Modules [3]).

7.11 ComM Services

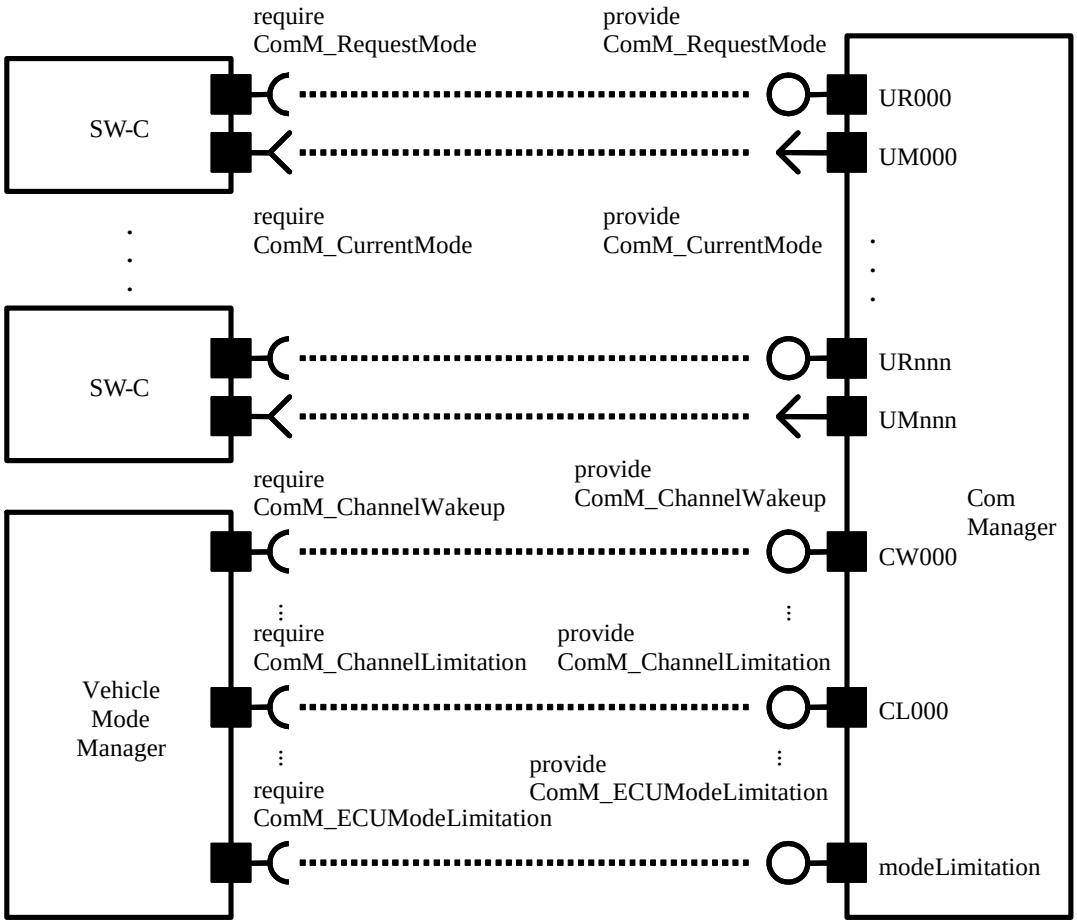
ComM698: This chapter defines the AUTOSAR Interfaces of the Communication Manager Service (COMM). The following definitions are interpreted to be in *ARPackage AUTOSAR/Services/ComM*.

7.11.1 Overview

7.11.1.1 Architecture

ComM701: The overall architecture of the ComManager service is depicted in the following picture.

ComM702:



ComM703: Figure 7.2: ARPackage ComM

7.11.1.2 Use Cases

7.11.1.2.1 SW-C does not care about the com-manager at all

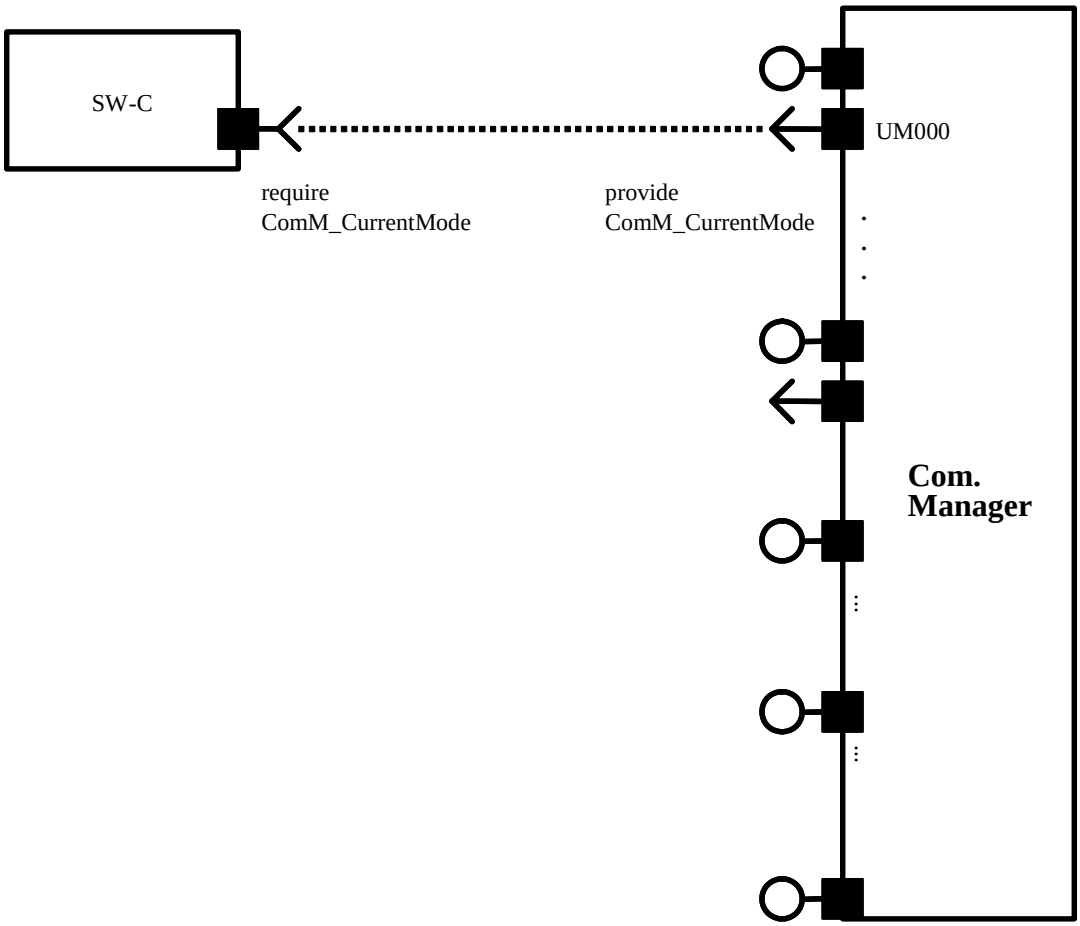
ComM705: A SW-C that does not care about the Communication Manager will not require any of the interfaces defined in the ARPackage ComM.

7.11.1.2.2 SW-C only cares about the state of its communication system

ComM708: In this use-case, a SW-C wants to know what communication capabilities it has (none, silent or full). It does this by defining a port requiring the Interface ComM_CurrentMode. Depending on the available

communication capabilities, the SW-C can specify that certain runnables of the SW-C should be executed or not. The ComManager must be configured correctly (with e.g. the physical channels that this SW-C uses for its logical communication) so that it has a port that provides this information about the current ComM_Mode to the SW-C.

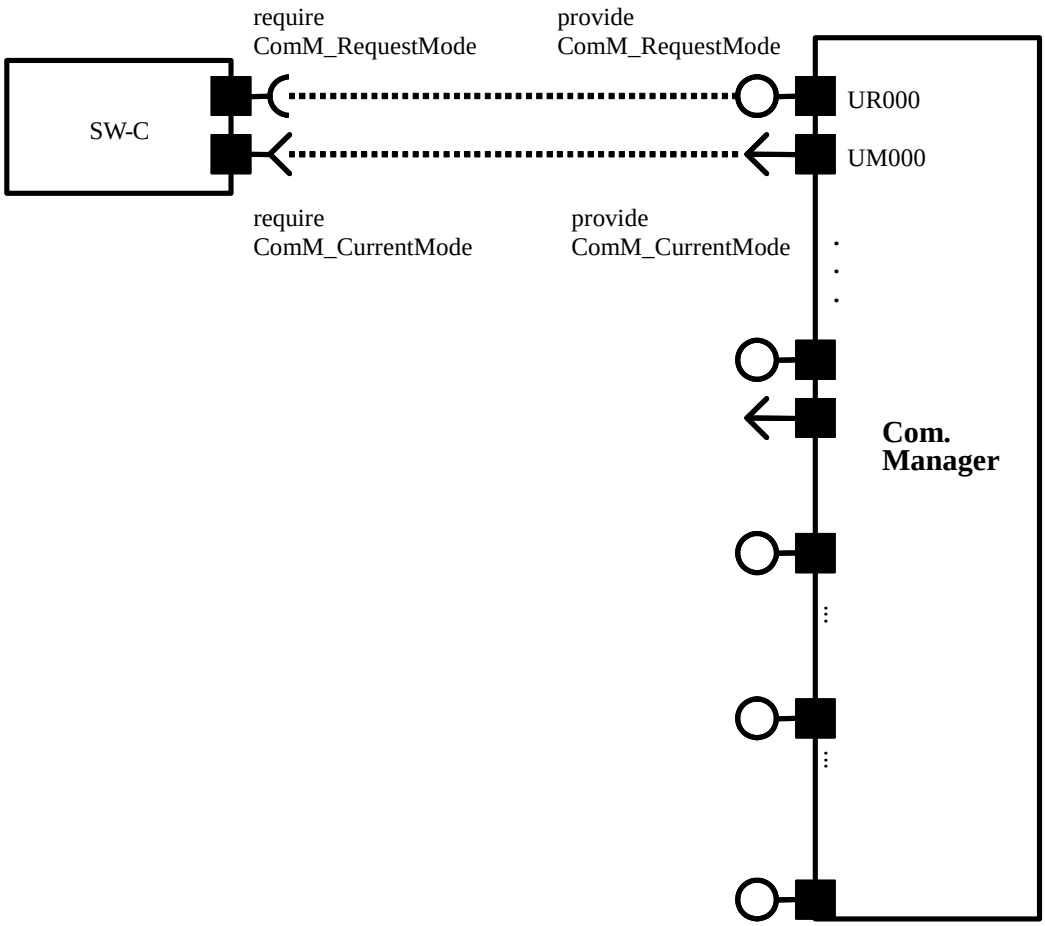
ComM709:



ComM710: Figure 7.3: SW-C requests state changes to ComM

7.11.1.2.3 SW-C explicitly wants to take influence on its communication state

ComM712:



ComM713: Figure 7.4: SW-C requires state changes within ComM and reads out current communication state

ComM715: In this case, the SW-C wants to explicitly take influence on the communication-state of the physical channels it needs. The SW-C indicates this by a port that requires the Interface *ComM_RequestComMode()*. Through this port, the SW-C can then request ComM Mode NO-COMMUNICATION or FULL-COMMUNICATION. The Communication Manager will use these calls to request the corresponding communication mode from the corresponding bus state manager. For a SW-C using the "direct API" of the RTE, the SW-C could for example do the following:

ComM714:
```
MySWC_Runnable_Init(self) {
// SW-C wants to send and receive data
e = Rte_Call_comRequest_RequestComMode(FULL_COMMUNICATION);
if (e == RTE_E_OK) {
// successfully requested the com-manager to move to
// full communication mode
} else {
// an error occurred when interacting with the com Manager
if (e == E_MODE_LIMITATION) {
```

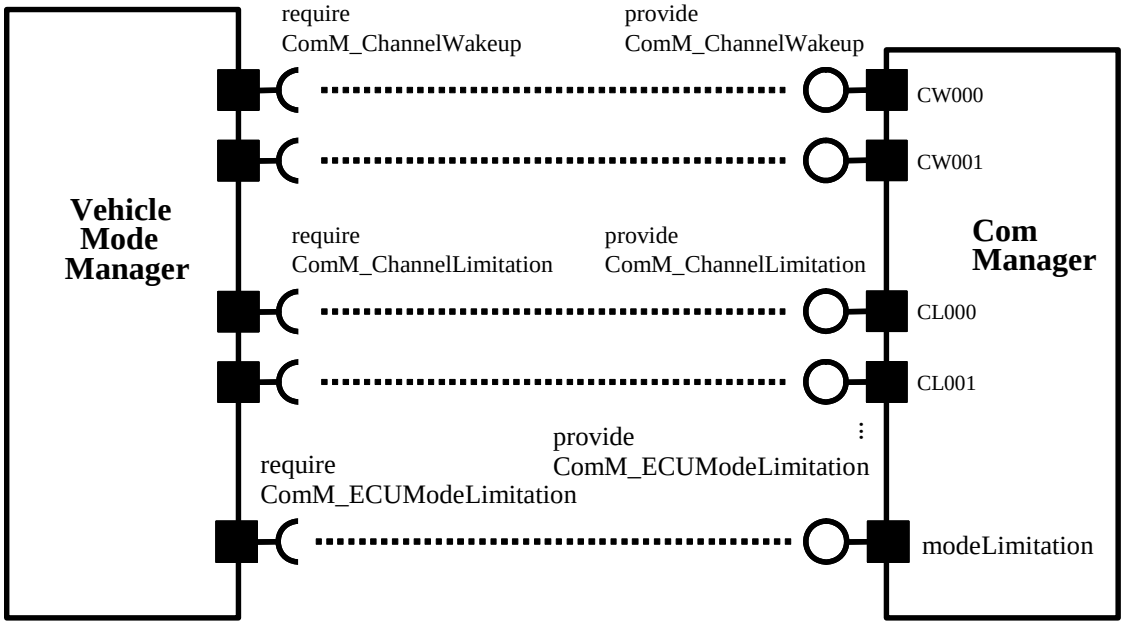
```
// a current ComMMode limitation forbids going into
// that mode; let's ask what the maximal allowed ComMMode is
Rte_Call_comRequest_GetMaxComMode(&max);
if(max==NO_COMMUNICATION) {
                …
};
                } else {
                        // a more serious error occurred …
                };
        };
        …
};

MySWC_Runnable_Loop(self) {
        if (status == ready_to_sleep) {
//no need to send; ready for shutdown communication
Rte_Call_comRequest_RequestComMode(NO_COMMUNICATION);
                …
        };
};
```

Comment: Note that these APIs do not require that the SW-C has knowledge of the channels that it needs.

7.11.1.2.4 SW-C wants to interact directly with physical channels activate ECU Mode Limitation

ComM717: This use-case will only occur when the SW-C has the role of a "vehicle state manager". There will typically only be one such component on an ECU.

ComM718:



ComM719: Figure 7.5: Interaction between VMM and ComM

7.11.1.3 Specification of Ports and Port Interfaces

ComM721: This chapter specifies the Port Interfaces that are needed in order to operate the COM Manager functionality over the RTE.

7.11.1.3.1 Types used by the interfaces

ComM760: PrimitiveTypeWithSemantics ComM_ModeType {
IntegerType {LOWER-LIMIT=0, UPPER-LIMIT=2};
0 -> NO_COMMUNICATION
1 -> SILENT_COMMUNICATION
2 -> FULL_COMMUNICATION
};

ComM761: ModeDeclarationGroup ComMMode {
{ NO_COMMUNICATION,
SILENT_COMMUNICATION,
FULL_COMMUNICATION }
initialMode = NO_COMMUNICATION
}

ComM762: OpaqueType InhibitionStatusType {
numberOfBits = 3
//bit 0: wakeup inhibition active
//bit 1: limit to “silent comm”
};

7.11.1.3.2 Ports and Port Interface for User Requests

7.11.1.3.2.1 General Approach

ComM724: A SW-C that wants to explicitly direct the local communication manager of the ECU towards a certain state requires the client-server interface ComM_UserRequest. Through this interface the SW-C can set the desired state of all communication channels that are relevant for that component, to “No Communication” or “Full Communication”. In order to keep the SW-C’s code independent from the values of the handles that are used to identify the user towards the ComM, these handles are not passed from the SW-C to the ComM. Rather they are modeled as “port defined argument values” of the Provide Ports on the ComM side. As a consequence, these handles do not show up as arguments in the operations of the client-server interface ComM_UserRequest. As a further consequence of this approach, the ComM has a separate port for each user.

7.11.1.3.2.2 Port interface ComM_UserRequest

ComM728: ClientServerInterface ComM_UserRequest
{

PossibleErrors {
E_NOT_OK = 1, //an internal execution error occurred
E_MODE_LIMITATION = 2 //ComMMode cannot be granted because of ComMMode inhibition
};

// the SW-C requests that all communication channels it needs are in the provided ComM Mode
RequestComMode(IN ComM_ModeType ComMode, ERR{E_NOT_OK,E_MODE_LIMITATION});

// returns the current ComM Mode for the SW-C
GetCurrentComMode(OUT ComM_ModeType ComMode, ERR{E_NOT_OK});

// returns the maximal allowed ComM Mode
GetMaxComMode(OUT ComM_ModeType ComMode, ERR{E_NOT_OK}):

// returns that last ComM Mode requested by the SW-C
GetRequestedMode(OUT ComM_ModeType ComMode, ERR{E_NOT_OK});
};

7.11.1.3.3 Ports and Port Interfaces for Current ComM Mode

7.11.1.3.3.1 General approach

ComM733: A SW-C that wants to get informed of its current ComM Mode shall require the sender-receiver interface ComM_CurrentMode. ComM needs to have a separate port providing this interface for each configured user.

7.11.1.3.3.2 Port interface

ComM735: SenderReceiverInterface ComM_CurrentMode {
ComMMode currentMode;
};

7.11.1.3.4 Ports and Port Interface for ECU Mode Limitation

7.11.1.3.4.1 General approach

ComM740: An implementation of ComM can be configured to provide the Interface ComM_ECUModeLimitation. A special SW-C playing the role of a “mode manager” can use this interface to change the behavior of the entire ECU.

7.11.1.3.4.2 Port interface

ComM742: ClientServerInterface ComM_ECUModeLimitation
{
PossibleErrors {
E_NOT_OK = 1 //an internal execution error occurred
};

//enables (status==true) or disables (status==false) the “no communication” ComM Mode
LimitECUToNoComMode(IN Boolean Status, ERR{E_NOT_OK}) ;

//returns the value of the “inhibited full communication request counter”
ReadInhibitCounter(OUT UInt16 CounterValue,ERR{E_NOT_OK});

//reset the “inhibited full communication request counter”
ResetInhibitCounter(ERR{E_NOT_OK});

//changes the ECU group classification status
SetECUGroupClassification(IN InhibitionStatusType Status,
ERR{E_NOT_OK});
};

7.11.1.3.5 Ports and Port Interface for Channel Wakeup

7.11.1.3.5.1 General approach

ComM747: An implementation of a ComM can be configured to provide the Interface ComM_ChannelWakeup. A special SW-C playing the role of a “mode manager” can use this interface to configure the ComM to take precautions against awaking other ECU's by starting the communication. In order to keep the SW-C’s code independent from the values of the handles that are used to identify a specific handle towards the ComM, these handles are **not** passed from the SW-C to the ComM. Rather they are modeled as “port defined argument values” of the Provide Ports on the ComM side. As a consequence, these handles do not show up as arguments in the operations of the client-server interface ComM_ChannelWakeup. As a further consequence of this approach, the ComM has separate ports for each channel.

7.11.1.3.5.2 Port interface

ComM749: ClientServerInterface ComM_ChannelWakeup {
PossibleErrors {
E_NOT_OK = 1 //an internal execution error occurred
};

//changes the inhibition status ComMNoWakeup for the channel
PreventWakeUp(IN Boolean Status, ERR{E_NOT_OK});
};

7.11.1.3.6 Ports and Port Interface for interface Channel Limitation

7.11.1.3.6.1 General approach

ComM752: An implementation of a ComM can be configured to provide the Interface ComM_ChannelLimitation. A special SW-C playing the role of a “mode manager” can use this interface to configure the ComM to inhibit communication mode for a given channel. In order to keep the SW-C’s code independent from the values of the handles that are used to identify a specific handle towards the ComM, these handles are **not** passed from the SW-C to the ComM. Rather they are modeled as “port defined argument values” of the Provide Ports on the ComM side. As a consequence, these handles do not show up as arguments in the operations of the client-server interface ComM_ChannelLimitation. As a further consequence of this approach, the ComM has separate ports for each channel.

7.11.1.3.6.2 Port interface

ComM756: ClientServerInterface ComM_ChannelLimitation {
PossibleErrors {
E_NOT_OK = 1 //an internal execution error occurred
};

//enables (status==TRUE) or disables (status==FALSE) the limitation of the channel to “no communication”
LimitChannelToNoComMode(IN Boolean Status, ERR{E_NOT_OK});

//returns the inhibition status of a channel
GetInhibitionStatus(OUT InhibitionStatusType Status, ERR{E_NOT_OK});
};

7.11.1.3.7 Definition of the Service ComM

ComM768: This section provides guidance on the definition of the ComM service. There are ports on both sides of the RTE. This description of the ComM service defines the ports below the RTE. Each SW-Component, which uses the Service, must contain “service ports” in its own SW-C description which will be connected to the ports of the COM Manager, so that the RTE can be generated.

Comment: Note that these definitions can only be completed during ECU configuration (because it depends on certain configuration parameters of the ComM which determine the number of ports provided by the ComM service). Also note that the implementation of a SW-C does *not* depend on these definitions.

ComM769: /* This is the definition of the ComM as a service. This is the “outside-view” of the ComM */

```
Service ComM {
//port present iff ComMModeLimitationEnabled
ProvidePort ComM_ECUModeLimitation modeLimitation;

//port present for each channel if
//ComMModeLimitationEnabled; there are NC channels;

ProvidePort ComM_ChannelLimitation CL000;
        …
ProvidePort ComM_ChannelLimitation CL<NC-1>;

//port present for each channel iff COMM_WAKEUP_INHIBITION_ENABLED
ProvidePort ComM_ChannelWakeup CW000;
        …
ProvidePort ComM_ChannelWakeup CW<NC-1>;

//For each user the ComM provides 2 ports.
//To facilitate configuration, the index of this user shall //correspond to the index in
        the array COMM_USER_LIST used for the
//configuration of the ComM (see ComM562).
//The number of users must correspond to the size of this array.
ProvidePort ComM_UserRequest UR000;
ProvidePort ComM_CurrentMode UM000;
ProvidePort ComM_UserRequest UR001;
ProvidePort ComM_CurrentMode UM001;
        …
ProvidePort ComM_UserRequest UR<COMM_USER_LIST.size-1>;
ProvidePort ComM_CurrentMode UM<COMM_USER_LIST.size-1>;
};
```

7.11.1.4 Runnables and Entry points

7.11.1.4.1 Internal behavior

ComM772: This is the inside description of the ComM. This detailed description is only needed for the configuration of the local RTE.

ComM773: InternalBehavior ComM {

```
        // Runnable entities of the ComM
        RunnableEntity LimitECUToNoComMode
            symbol “ComM_LimitECUToNoComMode”
```

```
                canbeInvokedConcurrently = FALSE
RunnableEntity ReadInhibitCounter
                symbol “ComM_ReadInhibitCounter”
                canbeInvokedConcurrently = FALSE
RunnableEntity ResetInhibitCounter
                symbol “ComM_ResetInhibitCounter”
                canbeInvokedConcurrently = FALSE
RunnableEntity SetECUGroupClassification
                symbol “ComM_SetECUGroupClassification”
                canbeInvokedConcurrently = FALSE
RunnableEntity LimitChannelToNoComMode
                symbol “ComM_LimitChannelToNoComMode”
                canbeInvokedConcurrently = FALSE
RunnableEntity GetInhibitionStatus
                symbol “ComM_GetInhibitionStatus”
                canbeInvokedConcurrently = FALSE
RunnableEntity PreventWakeup
                symbol “ComM_PreventWakeup”
                canbeInvokedConcurrently = FALSE
RunnableEntity RequestComMode
                symbol “ComM_RequestComMode”
                canbeInvokedConcurrently = TRUE
RunnableEntity GetMaxComMode
                symbol “ComM_GetMaxComMode”
                canbeInvokedConcurrently = TRUE
RunnableEntity GetRequestedMode
                symbol “ComM_GetRequestedMode”
                canbeInvokedConcurrently = TRUE
RunnableEntity GetCurrentComMode
                symbol “ComM_GetCurrentComMode”
                canbeInvokedConcurrently = TRUE

//the following applies iff ComMModeLimitationEnabled
modeLimitation.LimitECUToNoComMode -> LimitECUToNoComMode
modeLimitation.ReadInhibitCounter -> ReadInhibitCounter
modeLimitation.ResetInhibitCounter -> ResetInhibitCounter
modeLimitation.SetECUGroupClassification -> SetECUGroupClassification

//per-channel behavior only present if ComMModeLimitationEnabled //there are NC
        channels
//To facilitate configuration, the names of the channels correspond
//to the index of the channel in the “Channel” container used to
//configure the ComM

CL000.LimitChannelToNoComMode -> LimitChannelToNoComMode
CL000.GetInhibitionStatus -> GetInhibitionStatus
PortArgument {port=CL000, value.type=UInt8,
        value.value=Channel[0].COMM_CHANNEL_ID}
        …
```

CLnnn.LimitChannelToNoComMode -> LimitChannelToNoComMode
CLnnn.GetInhibitionStatus -> GetInhibitionStatus
PortArgument {port=CLnnn, value.type=UInt8,
value.value=Channel[nnn].COMM_CHANNEL_ID}

//per-channel behavior only present if COMM_WAKEUP_INHIBITION_ENABLED
CW000.preventWakeUp -> PreventWakeUp
PortArgument {port=CW000, value.type=UInt8,
value.value=Channel[0].COMM_CHANNEL_ID}
…
CWnnn.preventWakeUp -> PreventWakeUp
PortArgument {port=CWnnn, value.type=UInt8,
value.value=Channel[nnn].COMM_CHANNEL_ID}

//per-user behavior
//Note that the port-argument value must be consistent with the
//value in the configuration COMM_USER_LIST
//Note that the exact data-type of the UserHandleType must of course
//be defined BEFORE RTE_configuration, but does NOT affect the
//API seen by the SW-C's that use the service
UR000.RequestComMode -> RequestComMode
UR000.GetMaxComMode -> GetMaxComMode
UR000.GetRequestedMode -> GetRequestedMode
UR000.GetCurrentComMode -> GetCurrentComMode
PortArgument {port=UR000, value.type=UInt8, value.value=COMM_USER_LIST[0]}
…
URnnn.RequestComMode -> RequestComMode
URnnn.GetMaxComMode -> GetMaxComMode
URnnn.GetRequestedMode -> GetRequestedMode
URnnn.GetCurrentComMode -> GetCurrentComMode
PortArgument {port=URnnn, value.type=UInt8, value.value=COMM_USER_LIST[n]}
};

Comment: 'modeLimitation.LimitECUToNoComMode -> LimitECUToNoComMode' is supposed to define an OperationInvokedEvent that links the OperationPrototype to the runnable entity that is supposed to be executed.

7.11.1.4.2 Header file to be included by the ComM

ComM782: The RTE deals with the ComM as with any normal SW-C. The RTE will be able to generate a header-file based on the internal-behavior description of the ComM which contains for example a definition of the API's (like "Rte_Ports_CurrentMode_P") which are available to the ComM. This implies that an implementation of the ComM must include this generated header-file.

8 API specification

8.1 Imported types

8.1.1 Standard types

ComM518: ComM shall include the following types from Std_Types.h:

- Std_VersionInfoType
- boolean
- uint8
- Std_ReturnType

ComM692: ComM shall include NetworkHandleType from ComStack_Types.h.
ComM650: The Std_ReturnType shall be extended with the following defines.
ComM649:

#define	Value	Description
E_OK (already defined in Std_Types.h)	0	Function call has been successfully accomplished and returned.
E_NOT_OK (already defined in Std_Types.h)	1	Function call has been unsuccessfully accomplished and returned because of an internal execution error.
COMM_E_MODE_LIMITATION	2	Function call has been successfully but mode can not be granted because of mode inhibition.
COMM_UNINIT	3	ComM not initialized

8.2 Type definitions

8.2.1 ComM_InitStatusType

ComM494:

Name:	ComM_InitStatusType	
Type:	Enumeration	
Range:	COMM_UNINIT	The COM Manager is not initialized or not usable. This shall be the default value after reset. This status shall have the value 0.
	COMM_INIT	The COM Manager is initialized and usable.
Description:	Initialization status of ComM.	

8.2.2 ComM_InhibitionStatusType

ComM496:

Name:	ComM_InhibitionStatusType	
Type:	uint8	
Range:	0…7	Defines whether a mode inhibition is active or not. Bit 0 (LSB): Wake Up inhibition active Bit 1 (MSB): Limit to “No Communication” mode e.g. 00000011 -> Wake up inhibition and limitation to “No Communication” mode active
Description:	Inhibition status of ComM.	

8.2.3 ComM_UserHandleType

ComM246:

Name:	ComM_UserHandleType
Type:	Enumeration
Description:	Handle to identify a user. For each user, a unique value must be defined at system generation time.

Comment: This handle has local scope for only one ECU.

8.2.4 ComM_ModeType

ComM248:

Name:	ComM_ModeType	
Type:	Enumeration	
Range:	COMM_NO_COMMUNICATION	Communication Manager is in “No Communication” mode.
	COMM_SILENT_COMMUNICATION	Communication Manager is in “Silent Communication” mode.
	COMM_FULL_COMMUNICATION	Communication Manager is in “Full Communication” mode.
Description:	Current mode of the Communication Manager (main state of the state machine).	

8.3 Provided function definitions

8.3.1 ComM_Init

ComM146:

Service name:	ComM_Init
Syntax:	void ComM_Init( )
Service ID[hex]:	0x01
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	Initializes the AUTOSAR Communication Manager and restarts the internal state machines.

ComM793: Caveats: This function is called by the ECU State Manager. NVRAM manager have to be initialized to have the possibility to "direct" access the ComM parameters.

8.3.2 ComM_DeInit

ComM147:

Service name:	ComM_DeInit
Syntax:	void ComM_DeInit( )
Service ID[hex]:	0x02
Sync/Async:	Synchronous
Reentrancy:	Non Reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	De-initializes (terminates) the AUTOSAR Communication Manager.

ComM794: Caveats: This function is called by the ECU State Manager. De-init shall only be performed if all channels controlled by ComM are in "No Communication" state. The function call shall be ignored if the state is not "No Communication"

8.3.3 ComM_GetStatus

ComM242:

Service name:	ComM_GetStatus	
Syntax:	Std_ReturnType ComM_GetStatus(ComM_Init_StatusType* Status)	
Service ID[hex]:	0x03	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	Status	--
Return value:	Std_ReturnType	See Std_ReturnType
Description:	Returns the initialization status of the AUTOSAR Communication Manager.	

8.3.4 ComM_GetInhibitionStatus

ComM619:

Service name:	ComM_GetInhibitionStatus	
Syntax:	Std_ReturnType ComM_GetInhibitionStatus(NetworkHandleType Channel, ComM_InhibitionStatusType* Status)	
Service ID[hex]:	0x04	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	Channel	See NetworkHandleType
Parameters (inout):	None	
Parameters (out):	Status	See ComM_InhibitionStatusType
Return value:	Std_ReturnType	See Std_ReturnType
Description:	Returns the inhibition status of a ComM channel.	

8.3.5 ComM_RequestComMode

ComM110:

Service name:	ComM_RequestComMode	
Syntax:	Std_ReturnType ComM_RequestComMode(ComM_UserHandleType User, ComM_ModeType ComMode)	
Service ID[hex]:	0x05	
Sync/Async:	Synchronous	
Reentrancy:	reentrant	
Parameters (in):	User	Handle of the user who requests a mode
	ComMode	See ComM_ModeType
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	See Std_ReturnType
Description:	Requesting of a communication mode by a user. User Handle of the user who requests a mode. ComMode Name of the requested mode.	

ComM795: Configuration: Relationship between users and channels. A user is statically mapped to one or more channels.

8.3.6 ComM_GetMaxComMode

ComM85:

Service name:	ComM_GetMaxComMode	
Syntax:	Std_ReturnType ComM_GetMaxComMode(ComM_UserHandleType User, ComM_ModeType* ComMode)	
Service ID[hex]:	0x06	
Sync/Async:	Synchronous	
Reentrancy:	reentrant	
Parameters (in):	User	Handle of the user who requests a mode
Parameters (inout):	None	
Parameters (out):	ComMode	See ComM_ModeType
Return value:	Std_ReturnType	See Std_ReturnType
Description:	Function to query the maximum allowed communication mode of the corresponding user.	

Use Case: To have the possibility to request the maximum possible mode (e.g. user requests "Silent Communication" mode and wants to check if it is possible to get "Full Communication" mode or if a limitation/inhibition is active). Needed for diagnosis/debugging.

ComM374: If more than one channel is linked to one user request and the maximum allowed modes of the channels are different, the user gets always the lowest mode as a response to a mode request.

Comment: Sequence: No Communication -> Silent Communication -> Full Communication

ComM796: Configuration: Relationship between users and channels. A user is statically mapped to one or more channels.

8.3.7 ComM_GetRequestedComMode

ComM79:

Service name:	ComM_GetRequestedComMode	
Syntax:	Std_ReturnType ComM_GetRequestedComMode(ComM_UserHandleType User, ComM_ModeType* ComMode)	
Service ID[hex]:	0x07	
Sync/Async:	Synchronous	
Reentrancy:	reentrant	
Parameters (in):	User	Handle of the user who requests a mode
Parameters (inout):	None	
Parameters (out):	ComMode	Name of the requested mode
Return value:	Std_ReturnType	See Std_ReturnType
Description:	Function to query the currently requested communication mode of the corresponding user.	

Rationale: The requested user "Communication Mode" shall be stored volatile within the ComM itself, to prevent redundant storage of status information by the users.

Comment: If the ComM would not have this service every user has to store the status on its own --> redundant and maybe inconsistent storage of the same data.

ComM797: Configuration: Relationship between users and channels. A user is statically mapped to one or more channels.

8.3.8 ComM_GetCurrentComMode

ComM83:

Service name:	ComM_GetCurrentComMode	
Syntax:	Std_ReturnType ComM_GetCurrentComMode(ComM_UserHandleType User, ComM_ModeType* ComMode)	
Service ID[hex]:	0x08	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	User	Handle of the user who requests a mode
Parameters (inout):	None	
Parameters (out):	ComMode	See ComM_ModeType
Return value:	Std_ReturnType	See Std_ReturnType
Description:	Function to query the Current communication mode. ComM shall use the corresponding interfaces of the Bus State Managers to get the current	

	communication mode.

ComM176: If more than one channel is linked to one user request and the modes of the channels are different, the user shall get always the lowest mode as a response to a mode request.

Comment: Sequence: No Communication -> Silent Communication -> Full Communication

ComM798: Configuration: Relationship between users and channels. A user is statically mapped to one or more channels.

8.3.9 ComM_PreventWakeUp

ComM156:

Service name:	ComM_PreventWakeUp	
Syntax:	Std_ReturnType ComM_PreventWakeUp(NetworkHandleType Channel, boolean Status)	
Service ID[hex]:	0x09	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	Channel	See NetworkHandleType
	Status	False: Wake up inhibition is switched off True: Wake up inhibition is switched on
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	See Std_ReturnType
Description:	Changes the inhibition status ComMNoWakeup for the corresponding channel. Status False: Wake up inhibition is switched off True: Wake up inhibition is switched on	

ComM799: Configuration: Configurable with COMM_WAKEUP_INHIBITION_ENABLED (see chapter 10.2.2).

8.3.10 ComM_LimitChannelToNoComMode

ComM163:

Service name:	ComM_LimitChannelToNoComMode	
Syntax:	Std_ReturnType ComM_LimitChannelToNoComMode(NetworkHandleType Channel, boolean Status)	
Service ID[hex]:	0x0b	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	Channel	See NetworkHandleType
	Status	False: Limit channel X to “No Communication” disabled True: Limit channel X to “No Communication” enabled
Parameters (inout):	None	

Parameters (out):	None	
Return value:	Std_ReturnType	See Std_ReturnType
Description:	Changes the inhibition status to ComMNoCom for the corresponding channel. Status False: Limit channel X to “No Communication” disabled True: Limit channel X to “No Communication” enabled	

ComM800: Configuration: Configurable with ComMModeLimitationEnabled and (see chapter 10.2.3).

8.3.11 ComM_LimitECUToNoComMode

ComM124:

Service name:	ComM_LimitECUToNoComMode	
Syntax:	Std_ReturnType ComM_LimitECUToNoComMode(boolean Status)	
Service ID[hex]:	0x0c	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	Status	False: Limit ECU to “No Communication” disabled True: Limit ECU to “No Communication” enabled
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	See Std_ReturnType
Description:	Changes the inhibition status to ComMNoCom. Status False: Limit ECU to “No Communication” disabled True: Limit ECU to “No Communication” enabled	

ComM801: Configuration: Configurable with ComMModeLimitationEnabled and COMM_RESET_AFTER_FORCING_NO_COMM (see chapter 10.2.2).

8.3.12 ComM_ReadInhibitCounter

ComM224:

Service name:	ComM_ReadInhibitCounter	
Syntax:	Std_ReturnType ComM_ReadInhibitCounter(uint16* CounterValue)	
Service ID[hex]:	0x0d	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	CounterValue	amount of rejected “Full Communication” user requests
Return value:	Std_ReturnType	See Std_ReturnType
Description:	This function returns the amount of rejected “Full Communication” user requests. CounterValue amount of rejected “Full Communication” user requests	

ComM802: Configuration: Configurable with ComMModeLimitationEnabled (see chapter 10.2.2). Function will only be available if ComMModeLimitationEnabled (see chapter 10.2.2) is enabled

8.3.13 ComM_ResetInhibitCounter

ComM108:

Service name:	ComM_ResetInhibitCounter	
Syntax:	Std_ReturnType ComM_ResetInhibitCounter( )	
Service ID[hex]:	0x0e	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	See Std_ReturnType
Description:	This function resets the “Inhibited Full Communication Request Counter”.	

ComM803: Configuration: Configurable with ComMModeLimitationEnabled (see chapter 10.2.2). Function will only be available if ComMModeLimitationEnabled (see chapter 10.2.2) is enabled

8.3.14 ComM_SetECUGroupClassification

ComM552:

Service name:	ComM_SetECUGroupClassification	
Syntax:	Std_ReturnType ComM_SetECUGroupClassification(ComM_InhibitionStatusType Status)	
Service ID[hex]:	0x0f	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	Status	See ComM_InhibitionStatusType
Parameters (inout):	None	
Parameters (out):	None	
Return value:	Std_ReturnType	See Std_ReturnType
Description:	Changes the ECU Group Classification status (see chapter 10.2.2) status See ComM_InhibitionStatusType	

8.3.15 ComM_GetVersionInfo

ComM370:

Service name:	ComM_GetVersionInfo	
Syntax:	void ComM_GetVersionInfo(Std_VersionInfoType versioninfo)	
Service ID[hex]:	0x10	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	None	
Parameters (inout):	None	
Parameters (out):	versioninfo	See Std_VersionInfoType
Return value:	None	
Description:	This function returns the published information (for details refer to table 10.3)	

8.4 Provided indication functions (Call-back notifications)

ComM468: ComM shall provide the following indication functions.
ComM620: All the provided indication functions shall be implemented pre-compile time.

8.4.1 AUTOSAR Network Management Interface

8.4.1.1 ComM_Nm_NetworkStartIndication

ComM383:

Service name:	ComM_Nm_NetworkStartIndication	
Syntax:	void ComM_Nm_NetworkStartIndication(NetworkHandleType Channel)	
Service ID[hex]:	0x15	
Sync/Async:	Synchronous	
Reentrancy:	Non Reentrant	
Parameters (in):	Channel	See NetworkHandleType
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Indication that a NM-message has been received in the Bus Sleep Mode, what indicates that some nodes in the network have already entered the Network Mode.	

ComM804: Configuration: Mandatory.
ComM805: Caveats: The ComM is initialized correctly.

8.4.1.2 ComM_Nm_NetworkMode

ComM390:

Service name:	ComM_Nm_NetworkMode	
Syntax:	void ComM_Nm_NetworkMode(NetworkHandleType Channel)	
Service ID[hex]:	0x18	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Channel	See NetworkHandleType
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Notification that the network management has entered Network Mode.	

ComM806: Caveats: The ComM is initialized correctly.
ComM807: Configuration: Mandatory.

8.4.1.3 ComM_Nm_PrepareBusSleepMode

ComM391:

Service name:	ComM_Nm_PrepareBusSleepMode	
Syntax:	void ComM_Nm_PrepareBusSleepMode(NetworkHandleType Channel)	
Service ID[hex]:	0x19	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Channel	Channel
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Notification that the network management has entered Prepare Bus-Sleep Mode. Reemtramcy: Reentrant (but not for the same NM-Channel)	

ComM808: Caveats: The ComM is initialized correctly.
ComM809: Configuration: Mandatory.

8.4.1.4 ComM_Nm_BusSleepMode

ComM392:

Service name:	ComM_Nm_BusSleepMode	
Syntax:	void ComM_Nm_BusSleepMode(NetworkHandleType Channel)	
Service ID[hex]:	0x1a	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Channel	Channel
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Notification that the network management has entered Bus-Sleep Mode. This callback function should perform a transition of the hardware and transceiver to bus-sleep mode.	

ComM810: Caveats: The ComM is initialized correctly.
ComM811: Configuration: Mandatory.

8.4.1.5 ComM_Nm_RestartIndication

ComM792:

Service name:	ComM_Nm_RestartIndication	
Syntax:	void ComM_Nm_RestartIndication(NetworkHandleType Channel)	
Service ID[hex]:	0x1b	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Channel	Channel
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	If NmIf has started to shut down the coordinated busses, AND not all coordinated busses have indicated bus sleep state, AND on at least on one of the coordinated busses NM is restarted, THEN the NM Interface shall call the callback function ComM_Nm_RestartIndication with the nmNetworkHandle of the channels which have already indicated bus sleep state.	

ComM812: Caveats: The ComM is initialized correctly.
ComM813: Configuration: Mandatory.

8.4.2 AUTOSAR Diagnostic Communication Manager

8.4.2.1 ComM_DCM_ActiveDiagnostic

ComM362:

Service name:	ComM_DCM_ActiveDiagnostic
Syntax:	void ComM_DCM_ActiveDiagnostic( )
Service ID[hex]:	0x1f
Sync/Async:	Synchronous
Reentrancy:	reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	Indication of active diagnostic by the DCM.

8.4.2.2 ComM_DCM_InactiveDiagnostic

ComM364:

Service name:	ComM_DCM_InactiveDiagnostic
Syntax:	void ComM_DCM_InactiveDiagnostic( )
Service ID[hex]:	0x20
Sync/Async:	Synchronous
Reentrancy:	reentrant
Parameters (in):	None
Parameters (inout):	None
Parameters (out):	None
Return value:	None
Description:	Indication of inactive diagnostic by the DCM.

8.4.3 AUTOSAR ECU State Manager

8.4.3.1 ComM_EcuM_RunModeIndication

ComM406:

Service name:	ComM_EcuM_RunModeIndication	
Syntax:	void ComM_EcuM_RunModeIndication(NetworkHandleType channel)	
Service ID[hex]:	0x29	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	channel	Channel
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Indication that ECU State Manager has entered “Run Mode”.	

Rationale: Communication can only start in run mode. Switching to "run" mode may need some time thus it is not guaranteed that run mode is entered immediately after the request. EcuM stays in RUN if at least one requests run.

8.4.3.2 ComM_EcuM_WakeUpIndication

ComM275:

Service name:	ComM_EcuM_WakeUpIndication	
Syntax:	void ComM_EcuM_WakeUpIndication(NetworkHandleType Channel)	
Service ID[hex]:	0x2a	
Sync/Async:	Synchronous	
Reentrancy:	Reentrant	
Parameters (in):	Channel	Channel
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Notification of a wake up on the corresponding channel.	

ComM814: Caveats: The ComM is initialized correctly.
ComM815: Configuration: Mandatory.

8.4.4 Bus State Manager Interface

8.4.4.1 ComM_BusSM_ModeIndication

ComM675:

Service name:	ComM_BusSM_ModeIndication	
Syntax:	void ComM_BusSM_ModeIndication(NetworkHandleType Channel, ComM_ModeType* ComMode)	
Service ID[hex]:	0x33	
Sync/Async:	Asynchronous	
Reentrancy:	Reentrant	
Parameters (in):	Channel	See NetworkHandleType
	ComMode	See ComM_ModeType
Parameters (inout):	None	
Parameters (out):	None	
Return value:	None	
Description:	Indication of the actual bus mode by the corresponding Bus State Manager. ComM shall propagate the indicated state to the users with means of the RTE (see ComM661).	

ComM816: Caveats: The ComM is initialized correctly.
ComM817: Configuration: Mandatory.

8.5 Scheduled functions

8.5.1 ComM_MainFunction

ComM429:

Service name:	ComM_MainFunction_<Channel_Id>
Syntax:	void ComM_MainFunction_<Channel_Id>( )
Service ID[hex]:	0x60
Timing:	FIXED_CYCLIC
Description:	This function shall perform the processing of the AUTOSAR ComM activities that are not directly initiated by the calls e.g. from the RTE. There shall be one dedicated Main Function for each instance of ComM. Precondition: ComM shall be initialized

ComM818: Configuration: See chapter 10.2.2

8.6 Required functions

ComM632: An overview of the required interfaces is shown in figure 5.1.

8.6.1 Mandatory Interfaces

8.6.1.1 AUTOSAR NVRAM Manager

ComM103: ComM shall use the corresponding standardized services of the NVRAM manager for storing and reading non-volatile data. For details refer to the AUTOSAR NVRAM Manager Specification [7].

8.6.1.2 AUTOSAR ECU State Manager

ComM239: The ComM shall use the corresponding functions to request and release "Run Mode" of the ECU state Manager for the corresponding channel. For details refer to the AUTOSAR ECU State Manager Specification [6].

ComM660:

API function	*Module*	*Description*
EcuM_ComM_RequestRUN(NetworkHandleType channel)	EcuM	Request “run” mode for the corresponding channel from EcuM
EcuM_ComM_ReleaseRUN(NetworkHandleType channel)	EcuM	Release “run” mode for the corresponding channel from EcuM
boolean EcuM_ComM_HasRequestedRUN (NetworkHandleType channel)	EcuM	Query “run” status of EcuM for the corresponding channel from EcuM

8.6.1.3 AUTOSAR Network Management Interface

ComM261: The ComM shall use the corresponding functions to synchronize the bus start-up and shutdown of the Network Management. For details refer to the AUTOSAR NM Interface Specification [9].

ComM666:

API function	*Module*	*Description*
Nm_Init(const CNm_ConfigType * const cnmConfigPtr)	Nm	Initializes the NM modules
Nm_PassiveStartUp(const NetworkHandleType nmChannelHandle)	Nm	Request “passive startup” from NM
Nm_NetworkRequest(const NetworkHandleType nmChannelHandle)	Nm	Request to keep the bus awake from NM
Nm_NetworkRelease(const NetworkHandleType nmChannelHandle)	Nm	Release the request to keep the bus awake from NM

8.6.1.4 AUTOSAR Diagnostic Communication Manager

ComM266: The ComM shall use the corresponding functions (see ComM526) to control the communication capabilities of the DCM module.
Comment: DCM provides no functions to start/stop transmission and reception. DCM ensures to control communication according the indicated ComM states.
ComM172: The DCM shall implement the following callback routines:
ComM526:

API function	*Module*	*Description*
Dcm_ComM_NoComModeEntered(void)	Dcm	Indicating “No Communication Mode” to DCM
Dcm_ComM_SilentComModeEntered(void)	Dcm	Indicating “Silent Communication Mode” to DCM
Dcm_ComM_FullComModeEntered(void)	Dcm	Indicating “Full Communication Mode” to DCM

ComM693: If more than one channel is linked to one user request and the modes of the channels are different, ComM shall indicate always the lowest mode to Dcm.
ComM527: For details refer to the AUTOSAR DCM Specification [11].

8.6.1.5 AUTOSAR RTE interface provided by RTE to ComM for the SW-C

ComM91: ComM shall use the corresponding function provided by RTE to indicate modes to the users. There shall be one indication per user. Fan-out in case of a mode indication related to more than one user shall be done by ComM.
ComM663: If more than one channel is linked to one user request and the modes of the channels are different, the user shall get always the lowest mode indicated.
ComM662: The sequence of users shall start with user 0 up to user N and and the name of the mode ports shall be UM000, UM001, ... UM<N>.
Rationale: It shall be possible to use the port based API also to address specific users directly.
Comment: Within the array of ports, the ports are named alphabetically.
ComM778: ComM has the responsability to explicitly indicate changes in modes to each individual user. The ComM does this by calling the right API on the RTE through the ports “UMnnn”. There is one such port per configured user. An implementation of the ComM could use any of the normal RTE-mechanisms to signal changes in the mode to the users. Given the specific configurability of the ComM, using the RTE “Indirect API” seems most appropriate. This works as follows (consult the RTE specification for details).
ComM779: An implementation of the ComM can use the “Rte_Ports” API to obtain an array of the “UMnnn” ports at run-time.

```
/* Return an array of all ports that provide the interface ComM_CurrentMode.
   Because of the specific naming conventions chosen, the element n in this
   array of ports will reference to the port UM<nnn>. For example
   userModePorts[1] will be a handle on port UM001 */
userModePorts = Rte_Ports_ComM_CurrentMode_P();
```

The number of such userModePorts can be obtained through the call Rte_NPorts_ComM_CurrentMode_P. This value corresponds to the size of the COMM_USER_LIST array.

ComM780: To signal that a user n is in a new node, the ComM should: userModePorts[n].Switch_currentMode(newMode)

ComM661:

API function	*Module*	*Description*
Rte_Ports_UserMode_P()[n].Switch_currentMode (RTE_MODE_ComMode_FULL_COMMUNICATION)	RTE	Indicating “Full Communication Mode” to RTE
Rte_Ports_UserMode_P()[n].Switch_currentMode (RTE_MODE_ComMode_SILENT_COMMUNICATION)	RTE	Indicating “Silent Communication Mode” to RTE
Rte_Ports_UserMode_P()[n].Switch_currentMode (RTE_MODE_ComMode_NO_COMMUNICATION)	RTE	Indicating “No Communication Mode” to RTE

ComM525: For details refer to the AUTOSAR RTE specification [8] and AUTOSAR Services Mode Management specification[21].

8.6.1.6 Diagnostic Event Manager (Dem)

ComM634: ComM shall use the corresponding function provided by Dem to indicate Production errors. For details refer to the AUTOSAR Diagnostics Event Manager [15].

8.6.2 Optional Interfaces

8.6.2.1 AUTOSAR DET

ComM523:

API function	***Module***	***Description***	***Configuration parameter (description see chapter 10)***
Det_ReportError(ModuleId, InstanceId, ApiId, ErrorId)	Det	Development error notification	COMM_DEV_ERROR_DETECT

8.6.2.2 AUTOSAR CAN State Manager

ComM434: The ComM shall use the corresponding functions to control the communication capabilities of the CAN State Manager. For details refer to the AUTOSAR CAN State Manager Specification [24].

Comment: Those APIs can be called re-entrant, as long as different channel & controller numbers are used.

ComM678:

API function	*Module*	*Description*
CanSM_RequestComMode (NetworkHandle, ComM_Mode)	CanSm	Function to request a communication mode from the CAN State Manager.
CanSm_GetCurrentComMode (NetworkHandle, *ComM_ModePtr)	CanSm	Function to query the actual communication mode from the CAN State Manager.

8.6.2.3 AUTOSAR LIN State Manager

ComM168: The ComM shall use the corresponding functions to control the communication capabilities of the LIN State Manager. For details refer to the AUTOSAR LIN State Manager Specification [23].

ComM676:

API function	*Module*	*Description*
LinSM_RequestComMode (NetworkHandle, ComM_Mode)	LinSm	Function to request a communication mode from the LIN State Manager.
LinSM_GetCurrentComMode (NetworkHandle, *ComM_ModePtr)	LinSm	Function to query the actual communication mode from the LIN State Manager.

8.6.2.4 AUTOSAR FlexRay State Manager

ComM169: The ComM shall use the corresponding functions to control the communication capabilities of the FlexRay State Manager. For details refer to the AUTOSAR FlexRay State Manager Specification [25].

ComM677:

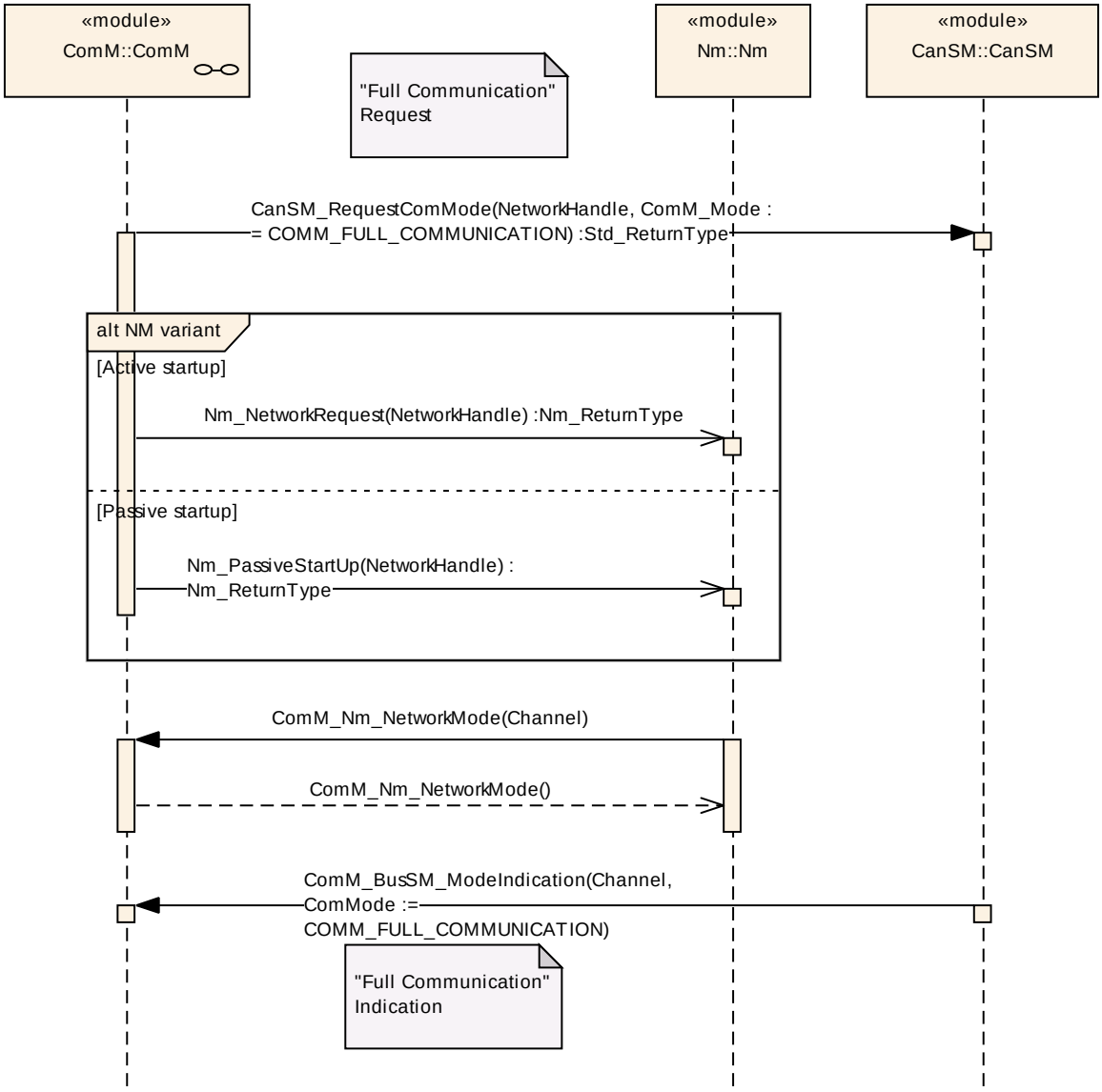
API function	*Module*	*Description*
FrSM_RequestComMode (NetworkHandle, ComM_Mode)	FrSm	Function to request a communication mode from the FlexRay State Manager.
FrSM_GetCurrentComMode (NetworkHandle, *ComM_ModePtr)	FrSm	Function to query the actual communication mode from the FlexRay State Manager.

9 Sequence diagrams

9.1 Transmission and Reception start (CAN)

ComM439: Figure 9.1 shows the sequence for starting transmission and reception on CAN. The behavior is equal for LIN and FlexRay just with different API names.

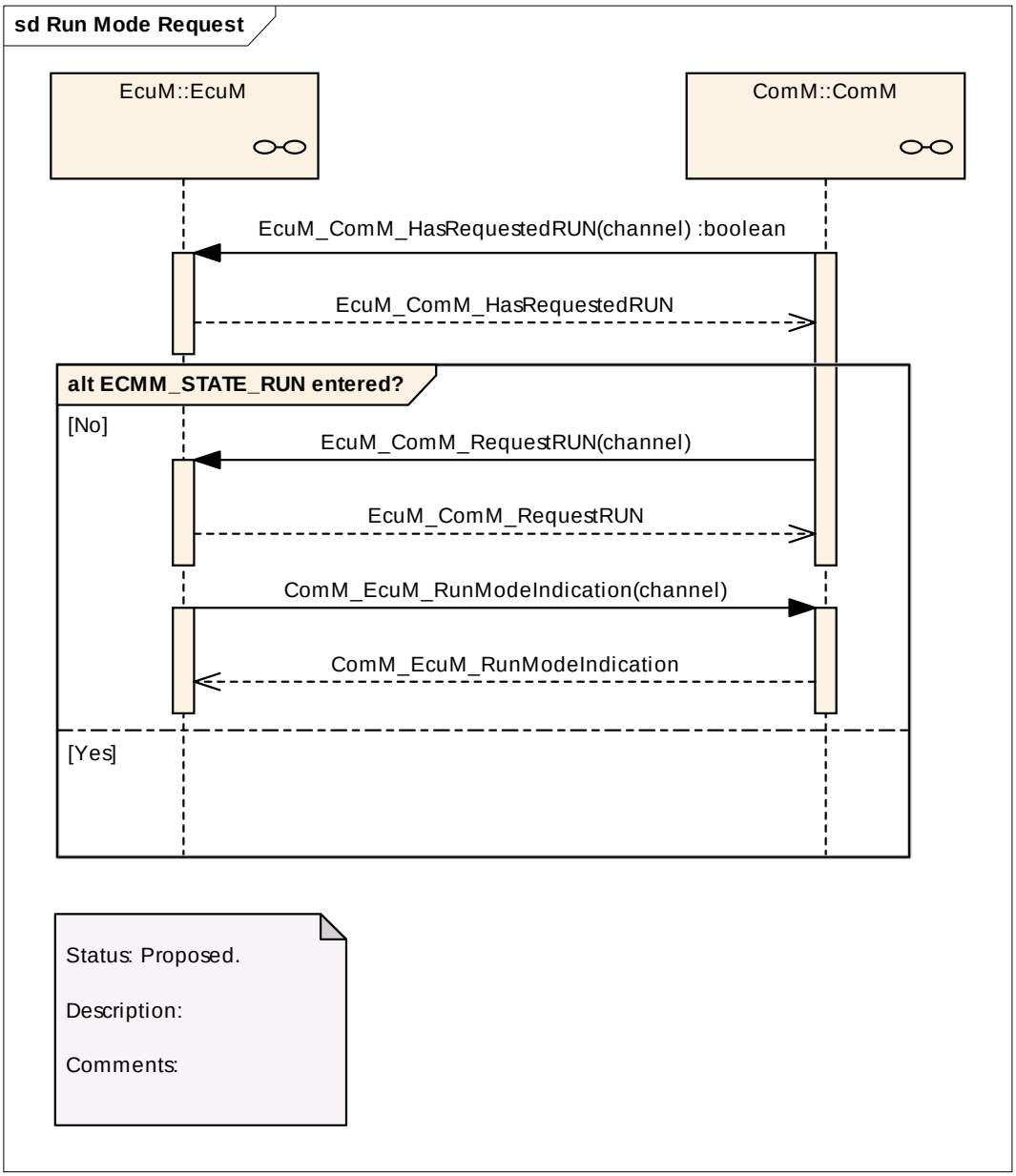
ComM441:



ComM442: Figure 9.1: Transmission and Reception start (CAN)

9.2 Run Mode request

ComM647: Figure 9.2 shows the behaviour of requesting "Run Mode" from EcuM.
ComM648: <Picture>

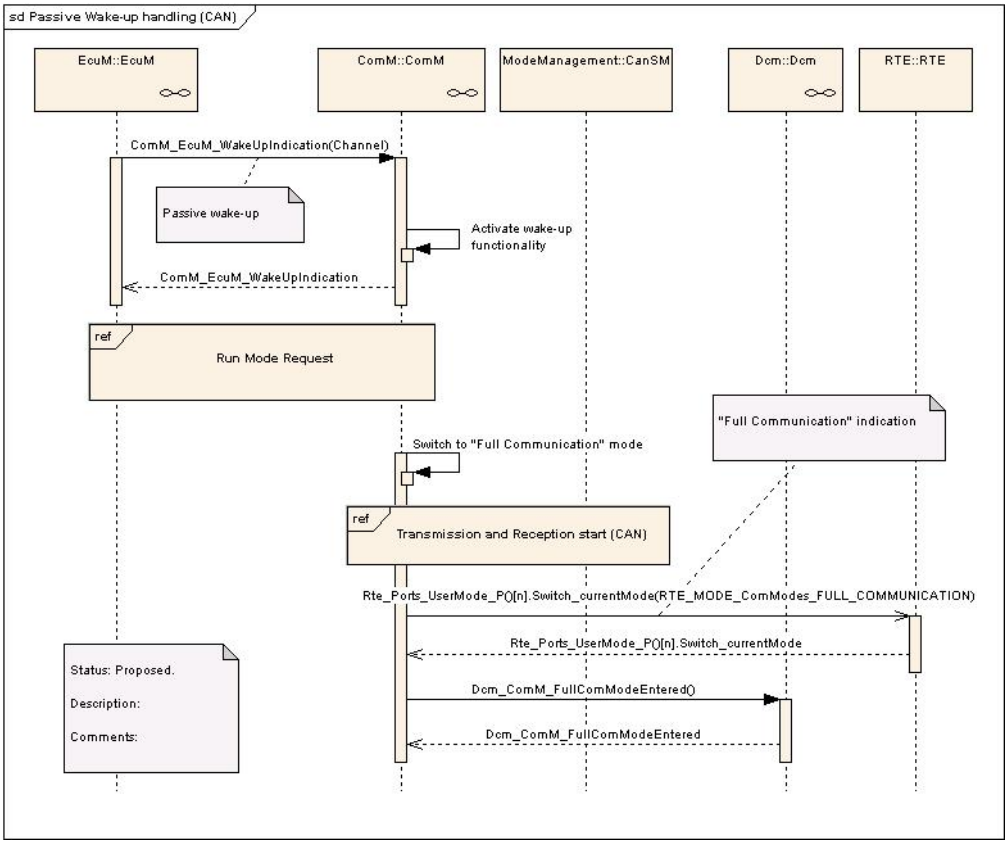


ComM651: Figure 9.2: Run Mode request

9.3 Passive Wake-up (CAN)

ComM316: Figure 9.3 shows the behaviour after a passive wake-up indicated by the ECU State Manager for a CAN channel. The behavior is equal for LIN and FlexRay just with different API names.

ComM33:

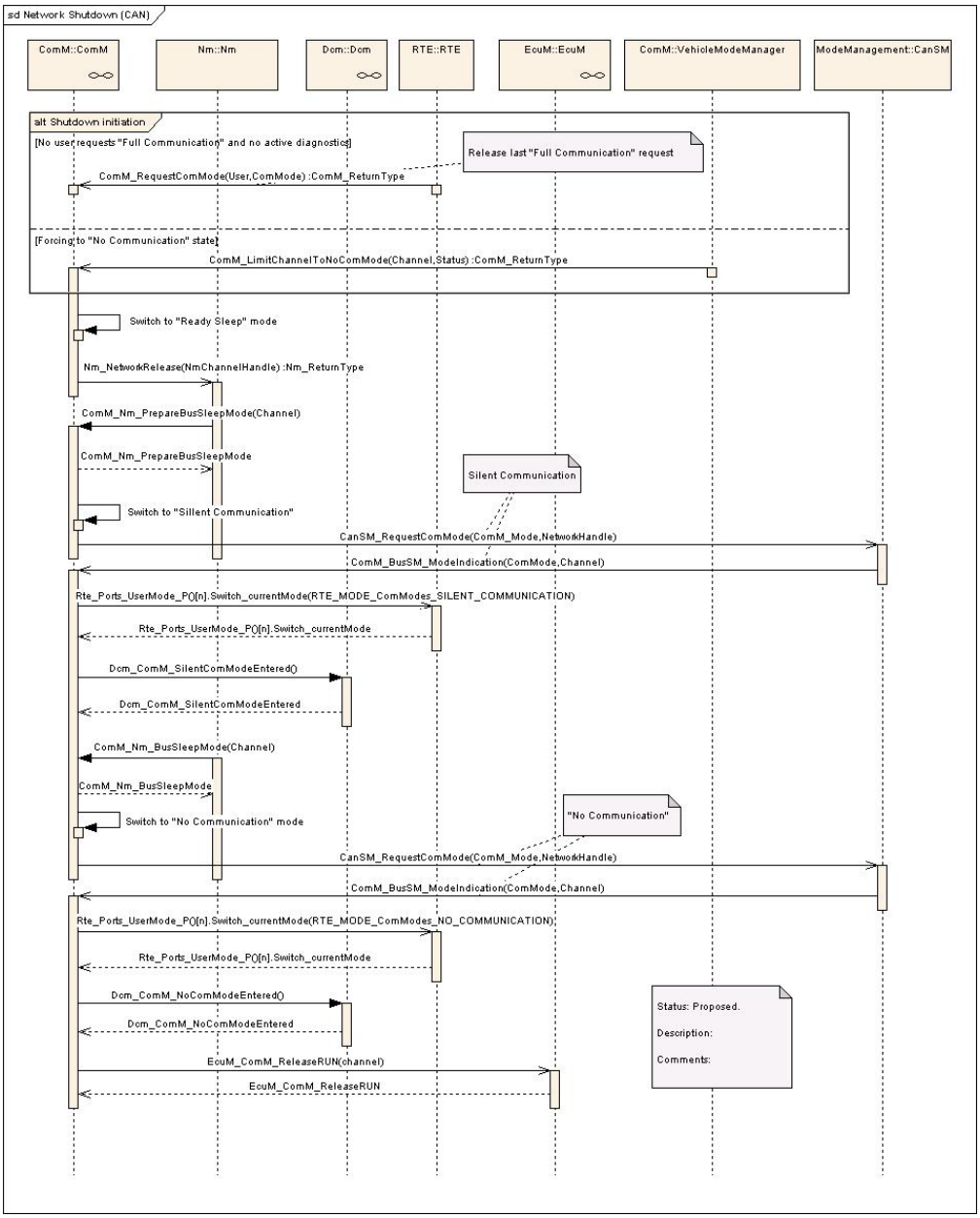


ComM317: Figure 9.3: Reaction on a passive wake-up indicated by the ECU State Manager

9.4 Network shutdown (CAN)

ComM318: Figure 9.4 shows the possibilities to shutdown the CAN network. It can be either initiated if the last user releases his "Full Communication" request or ComM_LimitChannelToNoComMode is called. The behavior is equal for LIN and FlexRay just with different API names.

ComM273:

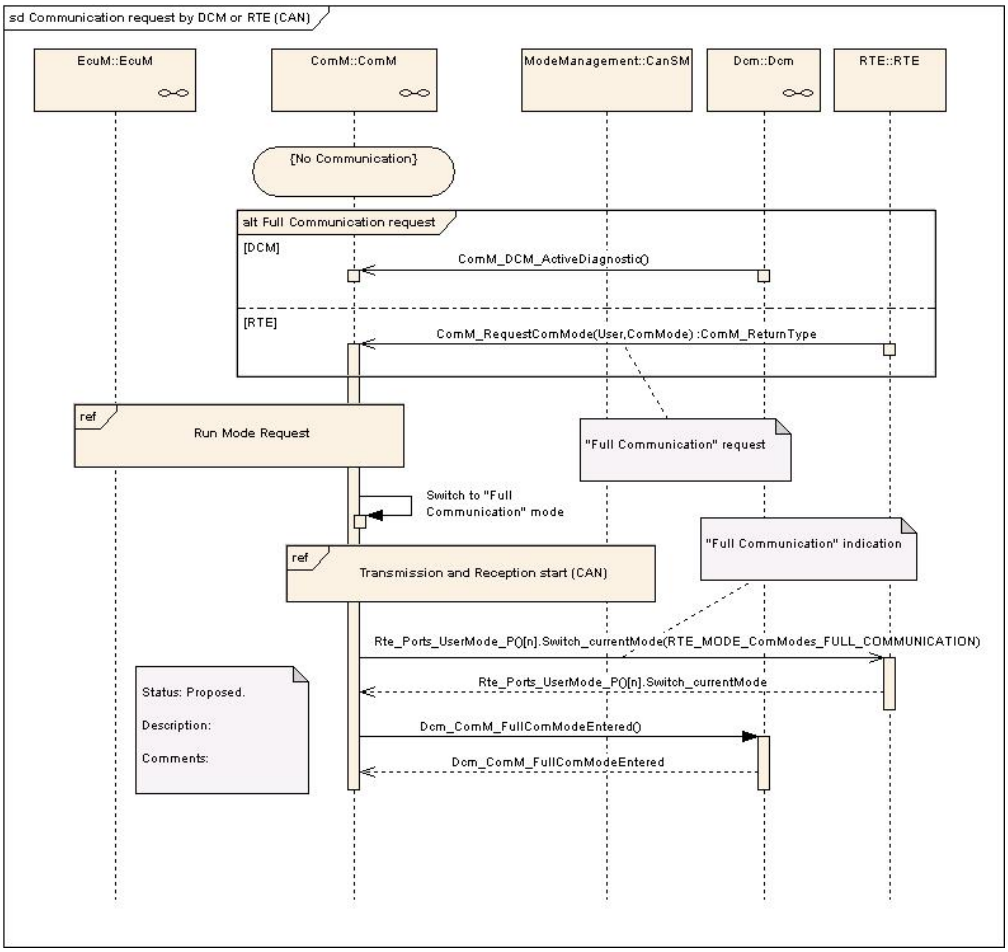


ComM319: Figure 9.4: Network shutdown (CAN)

9.5 Communication request

ComM453: Figure 9.5 shows the possibilities to start "Full Communication" on CAN. It can be either initiated if a user requests "Full Communication" request or DCM indicates ComM_DCM_ActiveDiagnostic. The behavior is equal for LIN and FlexRay just with different API names.

ComM454:



ComM455: Figure 9.5: Network start-up (CAN)

10 Configuration specification

ComM528: In general, this chapter defines configuration parameters and their clustering into containers. In order to support the specification Chapter 10.1 describes fundamentals.

Chapter 10.2 specifies the structure (containers) and the parameters of the module Communication Manager.

Chapter 10.3 specifies published information of the module Communication Manager.

10.1 How to read this chapter

ComM530: In addition to this section, it is highly recommended to read the documents:

- AUTOSAR Layered Software Architecture [2]
- AUTOSAR ECU Configuration Specification [5]

This document describes the AUTOSAR configuration methodology and the AUTOSAR configuration metamodel in detail.

The following is only a short survey of the topic and it will not replace the ECU Configuration Specification document.

10.1.1 Configuration and configuration parameters

ComM532: Configuration parameters define the variability of the generic part(s) of an implementation of a module. This means that only generic or configurable module implementation can be adapted to the environment (software/hardware) in use during system and/or ECU configuration.

The configuration of parameters can be achieved at different times during the software process: before compile time, before link time or after build time. In the following, the term “configuration class” (of a parameter) shall be used in order to refer to a specific configuration point in time.

10.1.2 Variants

ComM534: Variants describe sets of configuration parameters. E.g., variant 1: only pre-compile time configuration parameters; variant 2: mix of pre-compile- and post build time-configuration parameters. In one variant a parameter can only be of one configuration class.

10.1.3 Containers

ComM536: Containers structure the set of configuration parameters. This means

- *all* configuration parameters are kept in containers.

- (sub-) containers can reference (sub-) containers. It is possible to assign a multiplicity to these references. The multiplicity then defines the possible number of instances of the contained parameters.

10.1.4 Specification template for configuration parameters

ComM538: The following tables consist of three sections:
- the general section
- the configuration parameter section
- the section of included/referenced containers

ComM539:

SWS Item	*<[ReqXXX]>*
Container Name	*<Identifies the container by a name, e.g., CanDriverConfiguration>*
Description	*<Explains the intention and the content of the container .>*
Configuration Parameters	

ComM540:

Name	*<Identifies the parameter by name. The naming convention shall follow BSW00408.>*		
Description	*<Explains the intention of the configuration parameter.>*		
Type	*<Specify the type of the parameter (e.g., `uint8..uint32`) if possible or mark it "--">*		
Unit	*<Specify the unit of the parameter (e.g., `ms`) if possible or mark it "--" >*		
Range	*<Specify the range (or possible values) of the parameter (e.g., `1..15`, `ON, OFF`) if possible or mark it "--">*	*<Describe the value(s) or ranges.>*	
Configuration Class	***Pre-compile***	*see*	*<Refer here to (a) variant(s).>*
	Link time	*see*	*<Refer here to (a) variant(s).>*
	Post Build	*see*	*<Refer here to (a) variant(s).>*
Scope	*<Describe the scope of the parameter if known or mark it as "- -". The scope describes the impact of the configuration parameter: Does the setting affect only one instance of the module (instance), all instances of this module (module), the ECU or a network. Possible values of scope : instance, module, ECU, network>*		
Dependency	*<Describe the dependencies with respect to the scope if known ot mark it as "- -".>*		

ComM541:

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
<Reference a valid (sub)container by its name, e.g.,CanController>	*<Specifies the possible number of instances of the referenced container and its contained configuration parameters.* *Possible values: <multiplicity> <min_multiplicity.. max_multiplicity> >*	*<Describe the scope of the referenced sub-container if known or mark it as "- -".* *The scope describes the impact of the configuration parameter: Does the setting affect only one instance of the module (instance), all instances of this module (module), the ECU or a network.* *Possible values of scope :* *instance, module, ECU, network>* *<Describe the dependencies with respect to the scope if known ot mark it as "- -".>*

ComM542: Pre-compile time: Specifies whether the configuration parameter shall be of configuration class Pre-compile time or not.

ComM543:

Label	***Description***
x	The configuration parameter shall be of configuration class *Pre-compile time*.
--	The configuration parameter shall never be of configuration class *Pre-compile time*.

ComM544: Link time: Specifies whether the configuration parameter shall be of configuration class *Link time* or not.

ComM545:

Label	***Description***
x	The configuration parameter shall be of configuration class *Link time*.
--	The configuration parameter shall never be of configuration class *Link time*.

ComM546: Post Build: Specifies whether the configuration parameter shall be of configuration class *Post Build* or not

ComM547:

Label	***Description***
x	The configuration parameter shall be of configuration class *Post Build* and no specific implementation is required.
L	*Loadable* - the configuration parameter shall be of configuration class *Post Build* and only one configuration parameter set resides in the ECU.
M	*Multiple* - the configuration parameter shall be of configuration class *Post Build* and is selected out of a set of multiple parameters by passing a dedicated pointer to the init function of the module.
--	The configuration parameter shall never be of configuration class *Post Build*.

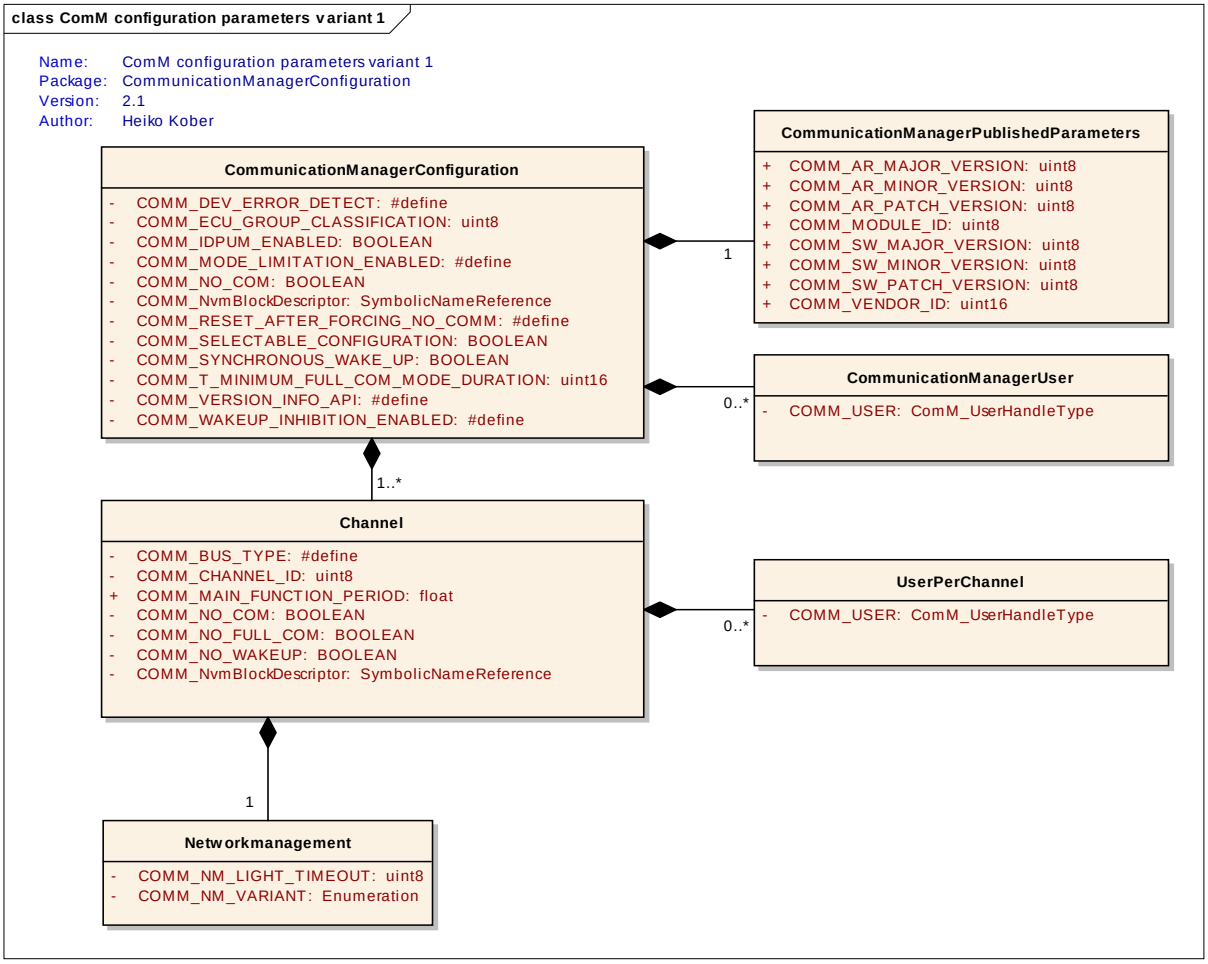
10.2 Containers and configuration parameters

ComM457: ComM configuration shall support a tool based configuration.
ComM419: Pre-compile time and link time configuration parameters shall be checked statically (at the latest during link time) for correctness.
ComM327: It shall be possible to assign communication-channels to users by static configuration.
ComM159: The relationship between users and channels shall be configurable.
Rationale: In a multi channel system each user can be assigned to one or more channels. If the user requests a mode, all channels assigned to this user, shall switch to the corresponding mode. All other channels shall not be affected.
ComM354: The DCM shall be related to all channels.
Rationale: DCM has no information about channels thus it don't know to which channel a diagnostic PDU is related.
ComM322: The bus type for each channel shall be configurable.
Rationale: Interfaces for controlling the communication stack depends on the bus type.
ComM464: ComM shall strictly separate configuration from implementation.
Rationale: Easy and clear configuration.
ComM456: Pre-compile time and published configuration data, shall group and export the configuration data to a static configuration interface. The name of the interface shall be ComM_Cfg.h.
ComM460: Files holding configuration data for ComM shall have a XML-format that is readable and understandable by human beings.

10.2.1 VARIANT-PRE-COMPILE

ComM549: ComM shall support only one variant called VARIANT-PRE-COMPILE. An overview of the parameters of VARIANT-PRE-COMPILE is shown in figure 10.1.

ComM585:



ComM584: Figure 10.1: Communication manager configuration parameters VARIANT-PRE-COMPILE

10.2.2 ComM

Module Name	*ComM*
Module Description	Configuration of the ComM (Communications Manager) module.

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
ComMChannel	1..*	This container contains the configuration (parameters) of the bus channel(s). The channel parameters shall be harmonized within the whole communication stack.
ComMGeneral	1	General configuration parameters of the Communication Manager.
ComMUser	1..*	This container contains a list of identifiers that are needed to refer to a user in the system which is designated to request Communication modes.

10.2.3 ComMGeneral

SWS Item	**ComM554 :**
Container Name	ComMGeneral{CommunicationManagerConfiguration}
Description	General configuration parameters of the Communication Manager.
Configuration Parameters	

SWS Item	**ComM555 :**		
Name	ComMDevErrorDetect {COMM_DEV_ERROR_DETECT}		
Description	Switches the Development Error Detection and Notification ON or OFF. true: Enabled false: Disabled		
Multiplicity	1		
Type	BooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**ComM563 :**		
Name	ComMEcuGroupClassification {COMM_ECU_GROUP_CLASSIFICATION}		
Description	Defines whether a mode inhibition affects the ECU or not. Examples: 000: No mode inhibition can be activated 001: Wake up inhibition can be enabled Forcing into "No Communication" mode shall be switched on if ComMNmVariant=PASSIVE.		
Multiplicity	1		
Type	IntegerParamDef		
Range	0 .. 255		
Default value	3		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module dependency: Shall be stored none volatile (value must be kept during a		

	reset.). Can be changed during runtime with ComM_SetECUGroupClassification() thus the default values shall be set only once (first ECU initialization).

SWS Item	**ComM560 :**		
Name	ComMModeLimitationEnabled {COMM_MODE_LIMITATION_ENABLED}		
Description	true if mode limitation functionality shall be enabled. true: Enabled false: Disabled		
Multiplicity	1		
Type	BooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module dependency: Shall be true if ComMNmVariant=COMM_PASSIVE		

SWS Item	**ComM561 :**		
Name	ComMNoCom {COMM_NO_COM}		
Description	The ECU is not allowed to change state of the ECU to "Silent Communication" or "Full Communication". true: Enabled (not allowed to switch to Silent Communication mode or Full Communication mode) false: Disabled		
Multiplicity	1		
Type	BooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module dependency: Shall not be stored none volatile (shall be reseted to default after initialization). Can be changed during runtime with ComM_LimitECUToNoComMode().		

SWS Item	**ComM558 :**		
Name	ComMResetAfterForcingNoComm {COMM_RESET_AFTER_FORCING_NO_COMM}		
Description	ComM shall perform a reset after entering "No Communication" mode because of an active mode limitation to "No Communication" mode. true: Enabled false: Disabled		
Multiplicity	1		
Type	BooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**ComM695 :**		
Name	ComMSynchronousWakeUp {COMM_SYNCHRONOUS_WAKE_UP}		
Description	Wake up of one channel shall lead to a wake up of all channels if true. true: Enabled false: Disabled		
Multiplicity	1		
Type	BooleanParamDef		
Default value	true		
ConfigurationClass	***Pre-compile time***	X	All Variants

	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: module		

SWS Item	**ComM557 :**		
Name	ComMTMinFullComModeDuration {COMM_T_MIN_FULL_COM_MODE_DURATION}		
Description	Minimum time duration in seconds, spent in the Full Communication mode.		
Multiplicity	1		
Type	FloatParamDef		
Range	0.0010 .. 65.0		
Default value	5.0		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**ComM622 :**		
Name	ComMVersionInfoApi {COMM_VERSION_INFO_API}		
Description	Switches the possibility to read the published information with the service ComM_GetPublishedInformation(). true: Enabled false: Disabled		
Multiplicity	1		
Type	BooleanParamDef		
Default value	true		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**ComM559 :**		
Name	ComMWakeupInhibitionEnabled {COMM_WAKEUP_INHIBITION_ENABLED}		
Description	true if wake up inhibition functionality enabled. true: Enabled false: Disabled		
Multiplicity	1		
Type	BooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module		

SWS Item	**ComM783 :**		
Name	ComMGlobalNvmBlockDescriptor {COMM_NvmBlockDescriptor}		
Description	Reference to NVRAM block containing the none volatile data		
Multiplicity	1		
Type	Reference to NvmBlockDescriptor		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: Derived from Nvm configuration		

No Included Containers

ComM689: ComMNoCom need not to be evaluated in case ComMModeLimitationEnabled is OFF thus it can be removed in that case to reduce/optimize the configuration.

10.2.4 ComMUser

SWS Item	**ComM653 :**
Container Name	ComMUser{CommunicationManagerUser}
Description	This container contains a list of identifiers that are needed to refer to a user in the system which is designated to request Communication modes.
Configuration Parameters	

SWS Item	**ComM654 :**		
Name	ComMUserIdentifier {COMM_USER}		
Description	An identifier that is needed to refer to a user in the system which is designated to request Communication modes. ImplementationType: ComM_UserHandleType		
Multiplicity	1		
Type	IntegerParamDef (Symbolic Name generated for this parameter)		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module dependency: EcuMUser: The concept of users is very similar to the concept of requestors in the ECU State Manager specification. These two parameters shall be harmonized during the configuration process.		

No Included Containers

10.2.5 ComMChannel

SWS Item	**ComM565 :**
Container Name	ComMChannel{Channel}
Description	This container contains the configuration (parameters) of the bus channel(s). The channel parameters shall be harmonized within the whole communication stack.
Configuration Parameters	

SWS Item	**ComM567 :**		
Name	ComMBusType {COMM_BUS_TYPE}		
Description	Identifies the bus type of the channel.		
Multiplicity	1		
Type	EnumerationParamDef		
Range	COMM_BUS_TYPE_CAN		
	COMM_BUS_TYPE_FR		
	COMM_BUS_TYPE_INTERNAL		
	COMM_BUS_TYPE_LIN		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**ComM635 :**
Name	ComMChannelId {COMM_CHANNEL_ID}

Description	Channel identification number of the corresponding channel.		
Multiplicity	1		
Type	IntegerParamDef (Symbolic Name generated for this parameter)		
Range	0 .. 255		
Default value	--		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: Shall be harmonized with channel IDs of networkmanagement and the bus interfaces.		

SWS Item	**ComM556 :**		
Name	ComMMainFunctionPeriod {COMM_MAIN_FUNCTION_PERIOD}		
Description	Specifies the period in seconds that the MainFunction has to be triggered with. Comment: ComM scheduling shall be at lest as fast as the communication stack and a schedule longer than 100ms makes no sense for communication.		
Multiplicity	1		
Type	FloatParamDef		
Range	0.0040 .. 0.1		
Default value	0.02		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU		

SWS Item	**ComM571 :**		
Name	ComMNoCom {COMM_NO_COM}		
Description	ECU is not allowed to change state of the channel to "Silent Communication" or "Full Communication". true: Enabled - Not allowed to switch to "Silent Communication mode" or "Full Communication mode" false: disabled This is the default/init value corresponding to a runtime variable that can be cahnged using ComM_LimitChannelToNoComMode().		
Multiplicity	1		
Type	BooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module dependency: ComMModeLimitationEnabled		

SWS Item	**ComM570 :**
Name	ComMNoFullCom {COMM_NO_FULL_COM}
Description	ECU is not allowed to change state of the channel to "Full Communication". TRUE - enabled - Not allowed to switch to "Full Communication mode" FALSE - disabled (default) This parameter is the default/init value of a corresponding runtime variable that can be changed using ComM_LimitChannelToNoComMode(). ComMNmVariant != FULL: ComM_LimitChannelToNoComMode() can be used. ComMNmVariant = PASSIVE: ComM_LimitChannelToNoComMode() shall not be used.
Multiplicity	1

Type	BooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module dependency: ComMNmVariant, ComMModeLimitationEnabled		

SWS Item	**ComM569 :**		
Name	ComMNoWakeup {COMM_NO_WAKEUP}		
Description	Defines if an ECU is not allowed to wake-up the channel. true: Enabled (not allowed to wake-up)) false: Disabled This is the default/init value of a runtime variable that can be changed during runtime using ComM_PreventWakeUp().		
Multiplicity	1		
Type	BooleanParamDef		
Default value	false		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module dependency: Shall be stored none volatile (value must be kept during a reset).		

SWS Item	**ComM670 :**		
Name	ComMNvmBlockDescriptor {COMM_NvmBlockDescriptor}		
Description	Reference to NVRAM block containing the none volatile data		
Multiplicity	1		
Type	Reference to NvmBlockDescriptor		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: ECU dependency: Derived from Nvm configuration		

Included Containers		
Container Name	***Multiplicity***	***Scope / Dependency***
ComMNetworkManagement	1	This container contains the configuration parameters of the networkmanagement.
ComMUserPerChannel	0..*	This container contains a list of identifiers that are needed to refer to a user in the system which is linked to a channel.

ComM691: ComMNoFullCom need not to be evaluated in case ComMModeLimitationEnabled is OFF thus it can be removed in that case to reduce/optimize the configuration.

ComM690: ComMNoCom need not to be evaluated in case ComMModeLimitationEnabled is OFF thus it can be removed in that case to reduce/optimize the configuration.

10.2.6 ComMNetworkManagement

SWS Item	**ComM607 :**
Container Name	ComMNetworkManagement{Networkmanagement}
Description	This container contains the configuration parameters of the networkmanagement.
Configuration Parameters	

SWS Item	**ComM606 :**		
Name	ComMNmLightTimeout {COMM_NM_LIGHT_TIMEOUT}		
Description	Defines the timeout (in seconds) after state "ready sleep" is left.		
Multiplicity	0..1		
Type	FloatParamDef		
Range	1.0 .. 255.0		
Default value	10.0		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module dependency: Only used if ComMNmVariant is configured as ComMLight		

SWS Item	**ComM568 :**		
Name	ComMNmVariant {COMM_NM_VARIANT}		
Description	Defines the functionality of the networkmanagement. Shall be harmonized with NM configuration.		
Multiplicity	1		
Type	EnumerationParamDef		
Range	FULL	AUTOSAR NM available (default).	
	LIGHT	No AUTOSAR NM available but functionality to shut down a channel.	
	NONE	No NM available	
	PASSIVE	AUTOSAR NM running in passive mode available.	
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency	scope: Module dependency: ComMNmVariant shall be NONE if ComMBusType = COMM_BUS_TYPE_INTERNAL		

No Included Containers

10.2.7 ComMUserPerChannel

SWS Item	**ComM657 :**
Container Name	ComMUserPerChannel{UserPerChannel}
Description	This container contains a list of identifiers that are needed to refer to a user in the system which is linked to a channel.
Configuration Parameters	

SWS Item	**ComM658 :**
Name	ComMUserChannel
Description	Reference to the ComMUser that corresponds to this channel user. ImplementationType: COMM_UserHandleType
Multiplicity	1

Type	Reference to ComMUser		
ConfigurationClass	***Pre-compile time***	X	All Variants
	Link time	--	
	Post-build time	--	
Scope / Dependency			

No Included Containers

10.3 Published parameters

ComM553: Published parameters contains data defined by the implementer of the SW module that does not change when the module is adapted (i.e. configured) to the actual HW/SW environment. It thus contains version and manufacturer information.
ComM324: The published parameters shown in table 10.2 shall be configurable.
ComM280:
ComM323: Table 10.2: Published parameters
ComM469: Enumeration of module version numbers shall be according the BSW General Requirements (For details refer to AUTOSAR General Requirements on Basic Software Modules [3])
ComM418: The version information in the module header and source files shall be validated and consistent (e.g. by comparing the version information in the module header and source files with a pre-processor macro).

The standard common published information like

vendorId (<Module>_VENDOR_ID),
moduleId (<Module>_MODULE_ID),
arMajorVersion (<Module>_AR_MAJOR_VERSION),
arMinorVersion (<Module>_ AR_MINOR_VERSION),
arPatchVersion (<Module>_ AR_PATCH_VERSION),
swMajorVersion (<Module>_SW_MAJOR_VERSION),
swMinorVersion (<Module>_ SW_MINOR_VERSION),
swPatchVersion (<Module>_ SW_PATCH_VERSION),
vendorApiInfix (<Module>_VENDOR_API_INFIX)

is provided in the BSW Module Description Template (see 3.1 Figure 4.1 and Figure 7.1).

Additional published parameters are listed below if applicable for this module.

11 Changes to Release 1

11.1 Deleted SWS Items

ComM617:

SWS Item	*Rationale*
ComM357, ComM423, 358-ComM360, ComM113, ComM496	Limitation to Silent Communication removed
ComM255, ComM394-ComM396, ComM399, ComM400, ComM427, ComM483, ComM636, ComM487	Bus off handling removed
ComM234	Timeout errors removed since timeout handling moves to Bus State Managers
ComM345, 250, ComM668, ComM669, ComM	Control of COM moved to Bus State Mangers
ComM642, ComM597	No longer initialized by ComM. Complete initialization by EcuM.
ComM67, ComM421, ComM638, ComM482, ComM425, ComM426, ComM403, ComM404, ComM645, ComM481, ComM573, ComM573, ComM613, ComM614	Functionality moved to Bus State Managers
ComM498, ComM574-ComM576	Functionality moved to CAN Bus State Manager
ComM356	Due to new architecture no longer possible to be performed by ComM. Have to be moved to Vehicle Mode Mangement.
ComM624, ComM314, ComM276, ComM430, ComM431, ComM643	ComM does no longer initialize the COM stack. Complete initialization by EcuM.
ComM451, ComM443, ComM452	Sub sequence chart incorporated in main sequence chart
ComM244, ComM660, ComM666	Bugzilla #18020
ComM564, ComM655, ComM578, ComM659, ComM605	Removed due to replacement of Chapter 10 by generated tables.
ComM639	Bugzilla #21855
ComM476	Bugzilla #21856

11.2 Replaced SWS Items

ComM683: None.

11.3 Changed SWS Items

ComM684:

SWS Item	*Rationale*
ComM463	Header files for Bus State Manager added
ComM52, ComM190, ComM151, ComM211	State machine revised due to new architecture (adding of Bus State Mangers)
ComM84, ComM92, ComM73, ComM128, ComM71, ComM72, ComM69, ComM402, ComM434, ComM394, ComM395, ComM674, ComM675, ComM434, ComM678, ComM168, ComM676, ComM169, ComM677, ComM441, ComM33, ComM273, ComM454, ComM628	New architecture due to adding of Bus State Managers thus communication state is requested from Bus State Manager instead of Bus Interface
ComM83	Communication Manager no longer stores the actual bus communication state.
ComM361, ComM563	Limitation to Silent Communication removed
ComM36, ComM50	Adapted to new architecture and initialization sequence (complete initialization by EcuM)
ComM234	Errors removed due to moving the functionality to the Bus State Managers
ComM146, ComM147, ComM40	ComM does no longer initialize the COM stack. Complete initialization by EcuM.
ComM191	Bugzilla #16253
ComM52	Bugzilla # 16258, #16267
ComM207	Bugzilla #16257
ComM637	Bugzilla #16262
ComM470	Bugzilla #16266
ComM649	Bugzilla #16276
ComM660, Com648	Bugzilla #16844
ComM463, ComM36	Bugzilla #12487
ComM147	Bugzilla #16280
ComM649	Bugzilla #16281
ComM156	Bugzilla #16285
ComM108	Bugzilla #16288
ComM224	Bugzilla #16287
ComM561, ComM563, ComM569, ComM570, ComM571	Bugzilla #16297, #17065
ComM582	Bugzilla #17414
ComM690-ComM692	Bugzilla #17501
ComM619, ComM156, ComM163, ComM383, ComM390, ComM391, ComM92, ComM406, ComM275, ComM395, ComM675	Bugzilla #18020
ComM311	Bugzilla #19254
ComM72	Bugzilla #18994
ComM210	Bugzilla #19301
ComM231	Bugzilla #19281
ComM242 revised	Bugzilla #19712
ComM675 revised	Bugzilla #20831
ComM52, ComM296, ComM207	Bugzilla #21002
ComM288, ComM133	*Nm_NetworkRelease()* is only available if COMM_NM_VARIANT = FULL is configured.
ComM554, ComM658	Changes due to replacement of Chapter 10 with generated tables.

11.4 Added SWS Items

ComM685:

SWS Item	*Rationale*
ComM348, ComM349, ComM353	New architecture due to adding of Bus State Managers
ComM671, ComM687	Missing behavior
ComM686	ComM283 split into 2 requirements
ComM688	FlexRay shutdown can not be interrupted to avoid partial networks.
ComM692	Bugzilla #18020
ComM693	Bugzilla #19264
ComM694, ComM695	Enable the possibility to wake up all channels if a wake up of one channel is indicated
Chapter 7.11	Bugzilla #19287 (ComM services)
ComM783	Bugzilla #19616
ComM792	Bugzilla #21002

12 Appendix

12.1 Implementation proposal for the extended functionality of chapter 7.2

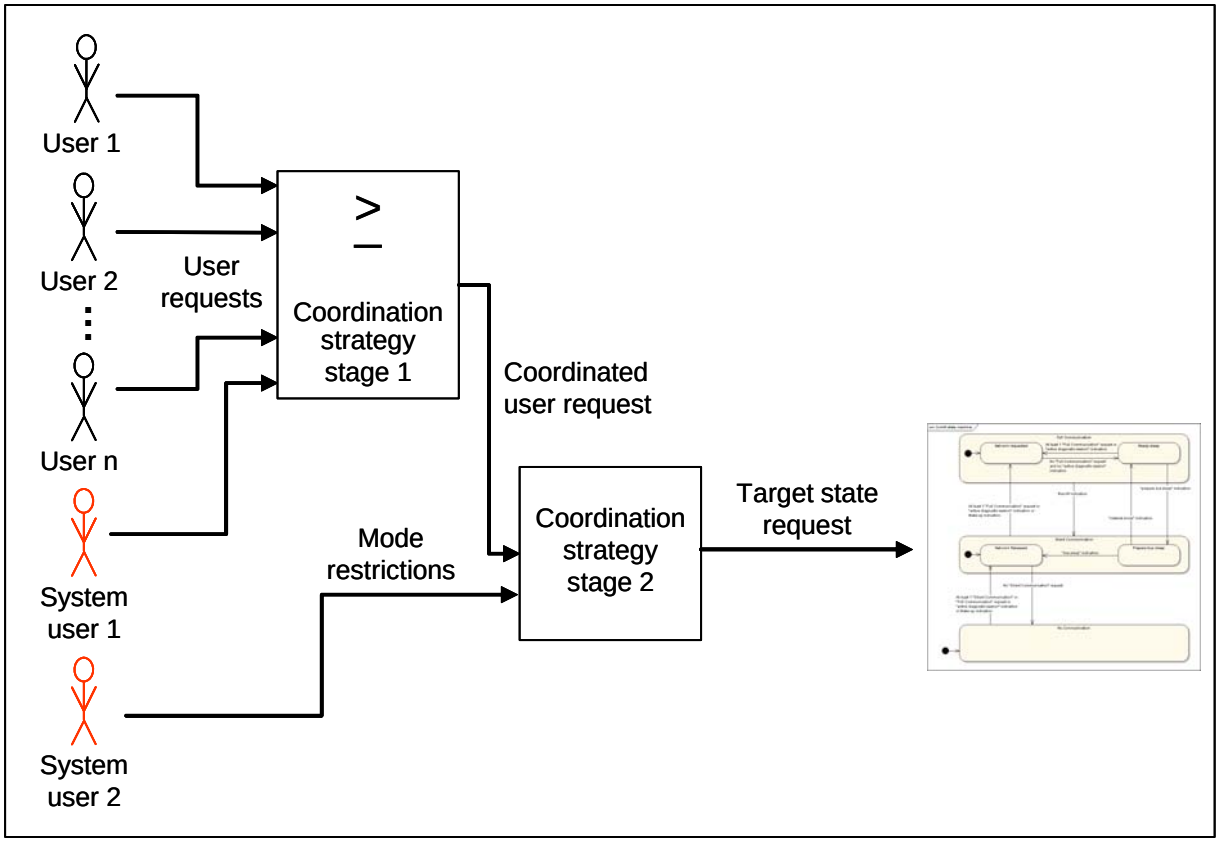
ComM61: The extended functionality can be implemented with two independent system users. System user 1 can be responsible for the state duration extensions and system user 2 can be responsible for the mode inhibition.

Comment: System users must be part of the ComM, invisible for other components.

12.1.1 User request coordination strategies

ComM112: The coordination strategy consists of the several stages shown in figure 12.1.

ComM114:



ComM115: Figure 12.1: User request coordination strategy overview

12.1.1.1 Coordination strategy stage 1

ComM416: If at least one user, including the system user 1 requests a communication mode for a channel and no other user including the system user 1 requests a higher communication mode, then the coordinated user

request for a channel will be a request to the state machine to enter the requested state for the channel.

Comment: Sequence: "no communication" -> "silent communication" -> "full communication"

12.1.1.2 Coordination strategy stage 2

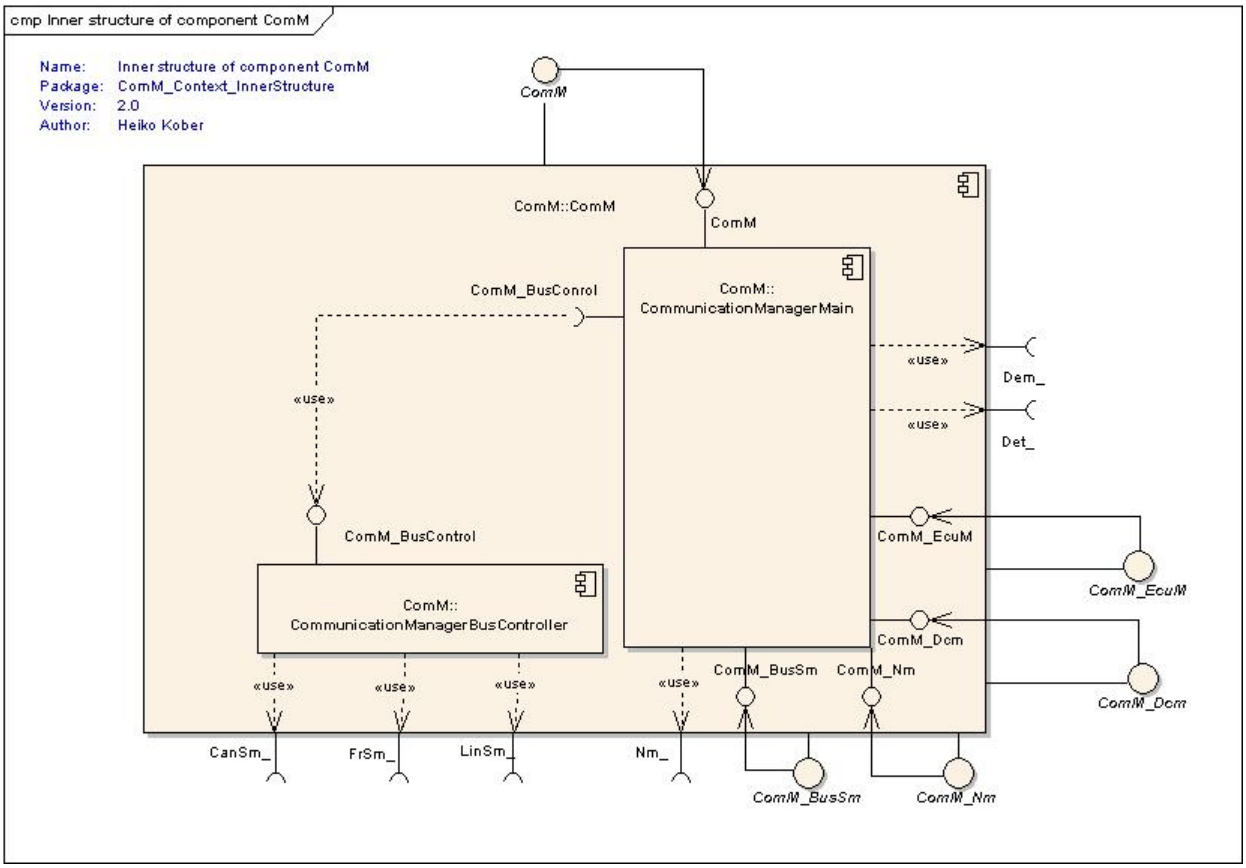
ComM417: Coordination strategy stage two checks whether a mode limitation is active or not. The coordinated user request from stage one will be inhibited in case of active mode limitations. E.g.:

- If the COM Inhibition status has a stored value of "ComMNoWakeup" for the corresponding channel a coordinated request for the Full Communication or Silent Communication state for channel X from Stage 1 is limited to the No Communication state for channel X.
- If the COM Inhibition status has a stored value of "ComMNoFullCom" for the corresponding channel a coordinated request for the Full Communication state for a channel from Stage 1 is limited to the Silent Communication state for channel X.
- If the COM Inhibition status has a stored value of "COM_NO_COM" all user requests for all channels are limited to the No Communication state.

12.2 Implementation proposal for the bus controlling

ComM627: Figure 12.2 shows a architecture proposal for an efficient bus controlling. The CommunicationManagerBusController component can be implemented as a macro (generated out of the configuration) and encapsulates the interface dependent APIs.

ComM628:



ComM629: Figure 12.2: Architecture proposal for an efficient bus controlling