Nm_Gmlan_Gm

Technical Reference

Version 2.03.01

Authors	Marco Pfalzgraf
Status	Released

1 Document Information

1.1 History

Author	Date	Version	Remarks
M. Radwick	2002-06-14	1.0	creation
M. Radwick	2002-12-24	1.1	Incorporate comments from Armin Happel. Added Introduction, Overview and Functional sections.
Klaus Emmert Ralf Fritz	2004-02-23	1.2	New Layout. Minor changes.
Ralf Fritz/ Laura Winder	2004-10-12	1.3	Minor changes in API chapter.
Ralf Fritz	2005-05-09	1.4	Data types changed
Ralf Fritz	2005-08-02	1.5	Macros to access the return value of IINwmIsActiveVN added
Ralf Fritz	2006-10-02	1.6	Changed description of bus-off recovery time.
Ralf Fritz	2007-03-23	1.7	Function description of IINwmGetActiveListVN changed. Calibration section removed Description of ApplNwmReinitRequest corrected.
Markus Schwarz	2007-12-06	2.00	ESCAN00021184 added description for GENy adapted to new template changed order of chapters
Markus Schwarz	2010-07-16	2.00.01	ESCAN0030766: added chapter 4.5
Marco Pfalzgraf	2012-08-15	2.01.00	ESCAN00055995, ESCAN00055998: adapted chapters 6.2 and 6.3.3
Marco Pfalzgraf	2012-08-31	2.02.00	ESCAN00054683: Corrected code example in chapter 5.3.2 ‘Periodic tasks’ ESCAN00060804: Added chapter 4.11
Marco Pfalzgraf	2012-10-26	2.02.01	Added chapter 2
Marco Pfalzgraf	2013-05-15	2.02.02	ESCAN00067275: Adapted description of callback ApplNwmReinitRequest
Marco Pfalzgraf	2015-01-19	2.03.00	ESCAN00080646: Added API description for context switch support
Marco Pfalzgraf	2015-12-18	2.03.01	ESCAN00069542: Adapted description about activation of init active VNs ESCAN00087111: Added limitation to IINwmTask API description and chapter 5.3.2.

Table 1-1 History of the Document

1.2 Reference Documents

No.	Source	Title	Version
[1]	GM	Communication Strategy Specification GMW 3104	1.5
[2]	GM	RSM Fault Detection and Mitigation Algorithm	-
[3]	GM	RSM GMLAN Handler Robustness Changes V2	-
[4]	GM	RSM GMLAN Handler NM Race Condition Resolution	-
[5]	Vector	Technical Reference GMLAN Calibration	2.01.00

Table 1-2 Reference Documents

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

Contents

Illustrations

Tables

2 Component History

This chapter describes the implementation history of the Vector Network Management for General Motors (since version 4.02.00).

2.1 Nm_Gmlan_Gm Version 4.02.00

In this version robustness changes were implemented according to [3].

2.1.1 What is new?

> The signal/node supervision timer is not started for a calibrateable time ‘Sleep Transition Time’ after VN activation.

> Additional ‘Sleep transition delay time’ was introduced as a calibrateable value.

Please refer to [5] ‘Technical Reference GMLAN Calibration’ for more information about these new calibrateable values.

2.1.2 What has changed?

> Initially active VNs are no more activated at power on. They are only activated by a High Level Voltage Wakeup (HLVW).

2.2 Nm_Gmlan_Gm Version 4.03.00

In this version robustness changes were implemented according to [2] and [4].

2.2.1 What is new?

> Introduced Fault Detection and Mitigation Algorithm (see chapter 4.11)

2.2.2 What has changed?

> Removed the possibility that the GMLAN handler enters a loop where it transmits a HLVW frame every 100ms (according to [4]).

3 Introduction

Nowadays cars are growing to become more and more complex systems. The functionality of a modern car is not dominated by mechanical components anymore. Electrical Control Units (ECU), sensors and actors became irreplaceable parts of a car. They are responsible for the reasonable functions of the power train, the chassis and the body of a car. An example for some ECUs is shown in Figure 3-1

In many ways the functionality of an ECU in a car depends on information provided by other ECUs. For example the ECU of the dashboard needs the number of revolutions per time of the wheels to display the car's speed. As a result communication between the ECUs is a significant component of a modern vehicle.

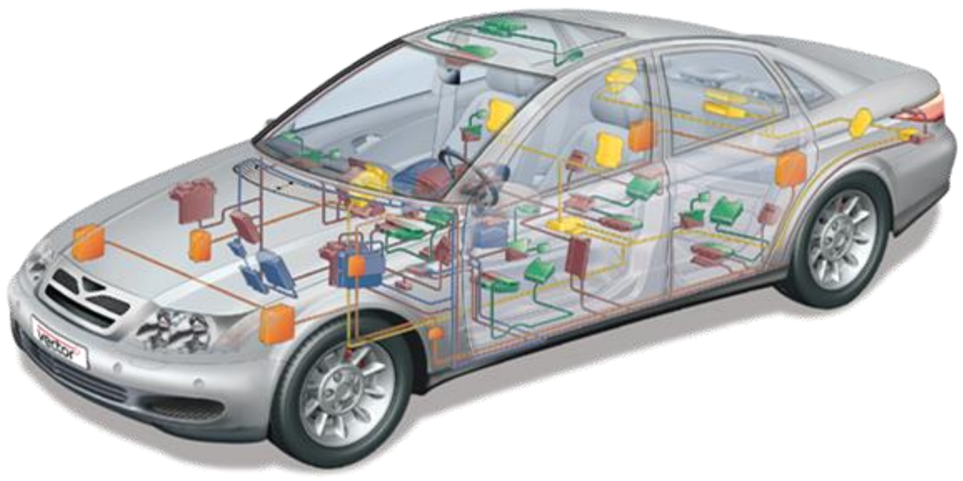


Figure 3-1 Example for Some ECU's in a Modern Car

The communication between ECUs should essentially remain encapsulated. The application working on an ECU should not need to know how to transmit or receive data from other ECUs. Therefore Vector Informatik GmbH provides a set of modules for the communication of ECUs by the CAN bus.

These communication modules are called CANbedded. They relieve the application of its communication assignment including the exchange of simple data, diagnostic data, NM data, calibration data and more. This document is concerned with how ECU's interact via NM.

3.1 Layer Concept

The implementation of the Network Management (NM) is intended to relieve the application of communication tasks. The NM is one of the communications modules of CANbedded offered by Vector Informatik GmbH. It is adapted to the specific requirements of General Motors. The CANbedded communication modules are organized in layers as shown in Figure 3-2. They consist of the Interaction Layer, the Network Management, the Transport Protocol, the Diagnosis Layer, the CAN Calibration Protocol and the CAN Driver (Data Link Layer).

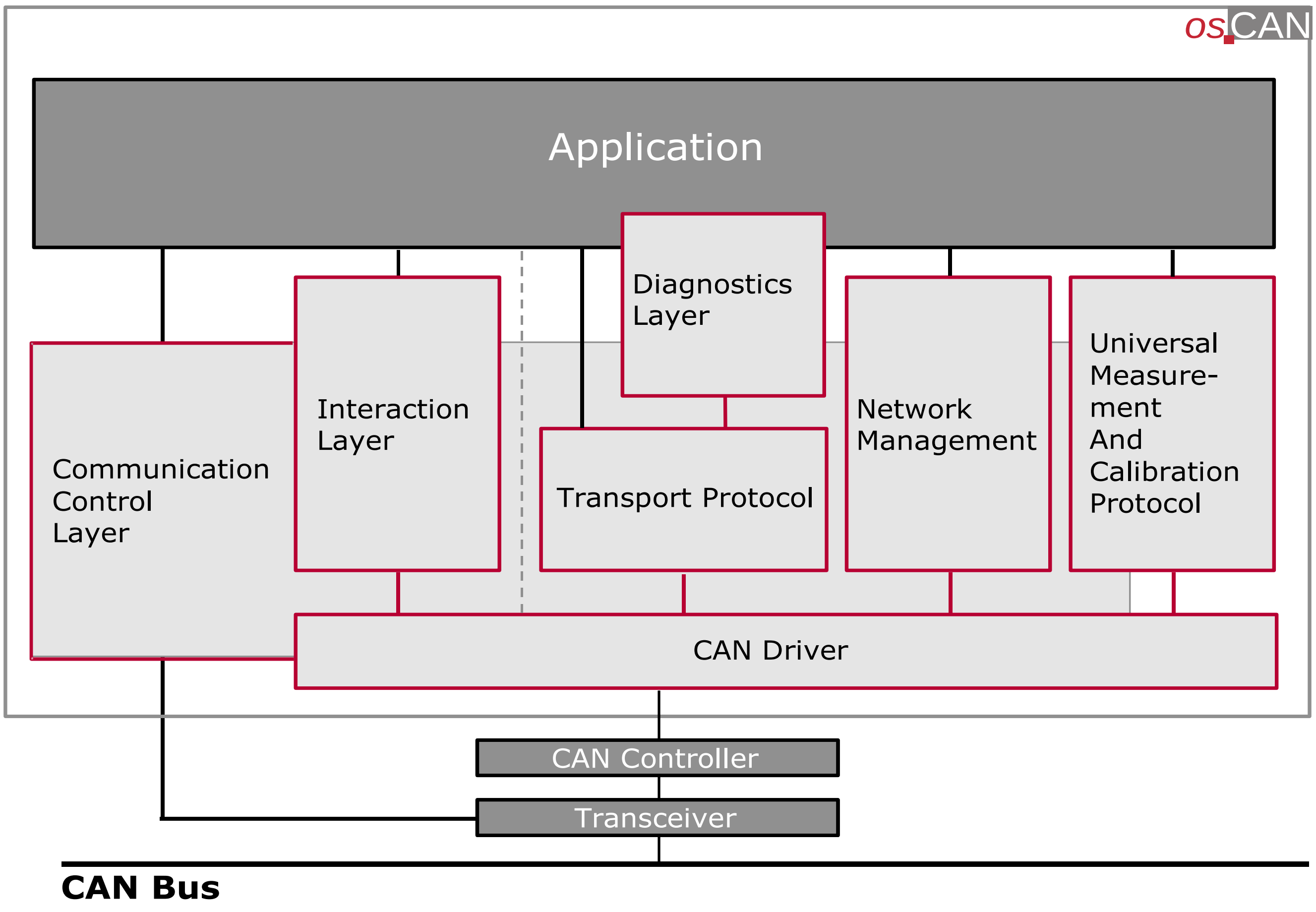


Figure 3-2 Layer model of Vector's CAN communication modules CANbedded

The availability of the CAN bus is controlled by the NM. The NM provides the following features:

> Control the start-up and the shut-down of the IL

> Control the activation and deactivation of VNs

> Control the peripheral hardware (CAN Controller and Bus Transceiver)

> Error recovery after BusOff

The Diagnosis Layer handles diagnostic services by CAN. It is used for the evaluation of the diagnosis requests and for the exception handling of invalid conditions like unknown services. To provide the diagnosis state, often rather long data, the Transport Protocol is used.

The CAN Calibration Protocol is specially designed for calibration and measurement data acquisition in ECUs. It has been defined by the European ASAP task force as a CAN based high speed interface for measurement and calibration systems (MCS).

If any information which has to be transmitted by the CAN bus does not fit into a single data frame because the data length exceeds 8 bytes, the Transport Protocol splits the data into several CAN messages using the same identifier.

The CAN Driver provides a mostly hardware independent interface to the higher communication layers. This enables the hardware independent implementation of the latter modules and the target platform independent reuse of them.

3.2 NM Features

GM's NM behavior is completely specified in [1].

The NM is used to control the start-up, shutdown, and error handling for the ECU. NM introduces the concept of a Virtual Network (VN), which is used by the system designer to divide the signals sent and received by an ECU into related functional groups. Use of VNs help conserve power and CAN bus bandwidth by permitting transmission and reception of only the signals and messages that are required at a given time. VNs may be individually active or inactive. The state of all VNs is communicated between ECUs using a Virtual Network Management Frame (VNMF). If a VN is active, then the ECU will be able to send and receive the signals associated with the VN. The NM will defer application requests to send a signal until one of its associated VNs is activated. If all the VNs of an ECU become inactive, then the ECU application is given the opportunity to go to sleep, thus conserving power.

The primary responsibilities of the NM are shown below:

> Keep VNs active on other nodes by sending out VNMF messages at fixed time intervals

> Activate relevant VNs upon receipt of VNMFs.

> Restart VN timer on reception and transmission of VNMF

> Count down the VN timer and deactivate VN when VN timer expires

> Respond to and recover from bus failures.

3.3 VN Concept

VNs are defined by the platform engineer to associate signals that are distributed among different ECUs in the CAN network. Every signal that may be exchanged between ECUs is associated with one or more VNs. The associations are defined using specific attributes in the message database. The purpose of this association is to minimize the number of messages being transmitted on the CAN bus at any given time. If there are no ECUs that require any signals associated with a VN, then the VN is deactivated, and transmission of those signals is halted.

ECUs are not required to participate in all VNs. The VNs an ECU participates in are determined by configuration settings given in the database. VN participation should be configured according to the CTS documentation released by GM.

There are four ways in which an ECU may be associated with a VN. The possible relationships are:

> Activator (Network Activated): The ECU, in response to some application related event, needs to send and/or receive signals. The application directs NM to activate one or more VNs. The ECU begins transmitting VNMF messages to notify other ECUs of the activation.

> Remotely Activated: The ECU is required respond to VN activations that are initiated by other ECUs. Remote activations are initiated in response to a received VNMF message.

> Shared Local: The ECU responds to an input event common to all modules that participate in the VN.

> Initially Active: The VN is temporarily activated by NM upon reception or transmission of a HLVW message.

Each VN may be configured as any combination of Activator, Remotely Activated, and Initially Active. However, if the VN is Shared Local, then the Activator and Remote options are excluded. The reason is related to how the activation is communicated to other ECUs. Normally, the NM will send a VNMF message on the CAN bus when an application requests that a VN be activated. Since ECUs participating in a Locally Activated VNs all see the same input event at the same time, there is no need to send or expect a VNMF message.

4 Functional Description

4.1 NM States

The behavior of VNs is defined for three primary states: COMM-OFF, COMM-ENABLED, and COMM-ACTIVE.

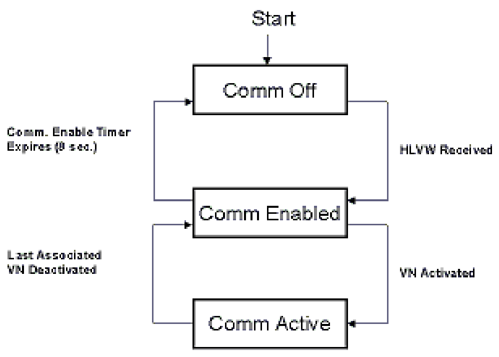

Start
Comm Off
Comm. Enable Timer Expires (8 sec.)
HLVW Received
Comm Enabled
Last Associated VN Deactivated
VN Activated
Comm Active

Figure 4-1 States of NM

COMM-OFF indicates that all VNs are deactivated, and that the CAN controller has been disabled. The application developer has the option to put the ECU micro-controller to sleep. During this state, no messages on the CAN bus can be processed. There are two ways to wake up the communications kernel: The application activates a VN, or, another ECU transmits a HLVW message. The NM responds to both of these events by entering the COMM-ENABLED state.

COMM-ENABLED is an intermediate state. While in this state, the communications kernel will process only one message: a VNMF. The VNMF message identifies all of the VNs that are remotely active. The NM examines the contents of a VNMF to determine if the ECU participates in any of the active VNs. If any relevant VNs are activated as a result of a VNMF message, or due to an application request, then NM will enter the COMM-ACTIVE state. If all relevant VNs remain inactive for a configured amount of time, the NM will return to the COMM-OFF state.

The NM will remain in the COMM-ACTIVE state so long as any VN that the ECU participates in is active. If the ECU application activates a (network-activated) VN, then NM will periodically transmit a VNMF message. The NM will continue sending VNMFs until the application deactivates all of the VNs that it started.

Reception of a VNMF is also used to keep VNs active. NM maintains a timer for each remotely activated VN. The timer for a VN is reset to a fixed value each time a VNMF message is received indicating that the VN is active. If the VNMF ceases to indicate that the VN is active (or if it ceases to arrive), then the VN timer(s) will eventually reach zero. When a timer reaches zero, NM will stop sending and receiving signals associated with the VN.

NM will transition from the COMM-ACTIVE state to the COMM-ENABLED state when it determines that all the VNs relevant to the ECU are inactive.

4.2 Normal Operation

At power-up, the NM will initialize all VNs as inactive. Afterwards, NM will enter the COMM-ENABLED state. After initialization, the application is free to activate any VN configured as Activator or Locally Active. To activate a VN, the application invokes `IlNwmActivateVN()`. Deactivation is accomplished using `IlNwmDeactivateVN()`.

Activation of a VN is affected by several factors. VNs configured as Activator or Locally-Activated are completely under the control of the application. VNs activated by the application will remain active until the application requests that the VN be deactivated. For Locally-Activated VNs, transmission and reception of signals is halted immediately. VNs configured as Activator are deactivated by NM 8 seconds after the application requests deactivation.

When the application requests activation of an Activator VN, NM will check to see how much time has elapsed since the last time a HLVW message has been sent. If the interval is too large, NM will automatically send a HLVW message in order to wake up all the ECUs on the network. NM will wait a short time (100ms) to give the other ECUs time to initialize, and then transmit a VNMF to notify the other ECUs of the VN activation.

NM will activate all VNs configured as Initially Active whenever a HLVW message is received. The VNs will remain active for 8 seconds and then automatically deactivate. If an Initially Active VN is already active when a HLVW message is received, the HLVW will reset the VN timers, allowing the Initially Active VNs to continue for 8 seconds after the HLVW message was received.

4.3 Low Voltage Tolerant Mode

Low Voltage Tolerant (LVT) mode is an optional feature intended to be used in situations when it is possible to predict when the ECU's voltage level will be low. In a low voltage environment, communication errors between ECUs will be more frequent. The purpose of LVT mode is to reduce the amount of time required to recover from the errors. LVT Mode is a distributed operation: all active ECUs should be programmed to enter LVT mode when a "LVT Master" ECU sends an entry request. The NM does not implement all the functions needed to support LVT mode. ECUs are required to implement application-specific behavior to completely support LVT mode. Please consult the CTS documentation for the ECU to determine if the application is required to support LVT mode.

When LVT mode is active, the NM will not transmit any HLVW messages. Only VNMFs needed to activate VNs will be transmitted (VNMFs needed to maintain active VNs will not be transmitted). In addition, the timers that control deactivation of VNs and signal supervision are disabled. As a result, active VNs will remain active until LVT mode is disabled.

LVT mode is the default at power-up. If CAN-OFF during Low-Voltage Mode is enabled, the application must exit Low Voltage mode before any messages can be transmitted.

Entry and exit from LVT mode is controlled by the ECU application via functions in the NM. LVT mode is automatically in effect when NM leaves the COMM-OFF state. This implies that the ECU responsible for LVT management (the "LVT Master") must exit LVT mode before activating the VN that carries the LVT exit signal. To enter LVT mode, call IlNwmEnterLowVoltageMode(). To enable transmission of HLVWs, call IlNwmExitLowVoltageMode().

The transmit path of the IL can also be disabled in LVT mode. The node will not send messages associated with active and activated VNs in that case. This feature can be enabled in the configuration.

LVT mode can be enabled in the configuration tool.

If enabled, the "CAN-Off during Low-Voltage Mode" option becomes available. The CAN-Off option modifies the behavior when LVT mode is activated by the application. If the CAN-Off option is selected, then ECU will stop sending all messages (instead of just HLVW messages). This implies that the application will be unable to activate any Network Activated VNs while LVT+CAN-Off mode is on.

4.4 High Load

Activation during High Load is another optional feature of NM. If the ECU configuration includes this, then the application will have the ability to inhibit (and restore) VN activation. If the application requests to activate a network activated VN (configured as Activator), then NM will defer activation until the application informs NM to allow VN activation. Requests to deactivate a deferred VN will clear the activation request. All outstanding activation requests will be attempted by NM when the application allows activation.

The High Load option is enabled in the configuration tool. To enable this option, select "VN Activation during high load" from the GMLAN Options tab. To inhibit VN activation, the application should invoke IlNwmInhibitActivationVN(), to restore activation, call IlNwmAllowActivationVN().

4.5 HighSpeed Mode

HighSpeed mode allows the ECU application to request an alternate communication speed on the CAN bus. This is normally used in response to a diagnostics request to download/flash calibration or program data.

Remote VN activation requests (via a received VNMF message) are ignored in HighSpeed Mode.

HighSpeed mode is intended to be used only with single-wire CAN networks.

HighSpeed mode is available only if enabled in the configuration

When HighSpeed mode is enabled, the configuration tool provides two CAN initialization objects (0=Standard, 1=HighSpeed). The CAN settings of these two initialization objects determine the used baudrate for each mode. These settings are used to configure the CAN controller hardware when a mode change occurs.

Typical baud rates are 33.333K for standard communication and 83.333K for HighSpeed communication.

HighSpeed mode should only be activated after the application has requested NormalCommunicationHalted mode.

To enter HighSpeed mode, the application calls IlNwmSetHispeedMode().

The standard communications rate is restored by invoking IlNwmResetHispeedMode().

4.6 Normal Communication Halted Mode

NormalCommunicationHalted (NCH) mode is used to support diagnostics communications. The application usually invokes this mode in response to a diagnostic service request (DisableNormalCommunications). When the application requests this mode, all VNs are deactivated. The diagnostics VN (VN 0) is activated. While in NCH mode, all requests to activate a VN are denied.

The application should invoke IlNwmNormalCommHalted() to halt normal communications. Calling IlNwmReturnToNormalMode() will restore normal communications. Note that all VNs remain deactivated after returning to normal mode. The application is responsible for re-activating any required VNs.

See also: GMW3110: GMLAN Enhanced Diagnostic Test Mode Specification

4.7 Bus Off

The CAN Data Link Layer specification requires that the CAN controller enters a BusOff state in the event of too many transmit errors. The NM is notified of this event by the CAN driver. In response to the first BusOff, the NM will re-initialize the CAN controller and restart communications. Upon restart, NM will start the BusOff Recovery Timer. If a subsequent BusOff event occurs before the timer expires, then the controller will be re-initialized, but the transmit path will remain disabled until the timer reaches zero. After enabling the transmit path, the NM will re-queue any messages pending transmission, and restart the recovery timer.

After recovering from the BusOff event by re-initialization of the controller, it is possible to receive signals from other ECUs.

If the application attempts to activate a Network Activated VN while NM is waiting to recover from BusOff, the activation request will be deferred. Upon recovery, the VN activation will attempted as normal.

If a remotely activated VN times out while waiting for BusOff recovery, then the VN will be deactivated as normal. Deactivation requests made by the application during BusOff will be granted, in which case messages associated with the VN that are pending transmission will be de-queued.

The application can be configured to be notified about a BusOff (ApplNwmBusoff()) and a BusOff recovery (ApplNwmBusoffEnd)().

The time required to recover from BusOff is also configurable. The value of “BusOff recovery time” defines the recovery time in milliseconds. The default value is 3500ms for body bus (single-wire) applications, and 110ms for powertrain (dual-wire) applications.

4.8 HLVW Failure Handling

On single-wire CAN networks, it is critical for the NM to confirm transmission of the HLVW message when activating a VN. NM will retry transmission of the HLVW each time the NM task is called for 100ms. If it fails, the CAN controller will be reset, and the activation is retried three times.

4.9 VN Activation Failure

The application may be notified in the event of VN activation failures. There are two notifications: VNMF Confirmation Timeout, and VN Activation Failed.

Activation of a Network Activated VN requires NM to transmit a VNMF to notify the other ECUs. If NM is unable to transmit the VNMF over a configured time-period, the application may be notified via the optional callback ApplNwmVnmfConfirmationTimeout(). The timeout time is determined by the value (in milliseconds) of the *“VNMF confirmation time”*.

In addition to the VNMF Confirmation Timeout, applications may select to be notified when individual VNs fail to activate. This feature is configurable.

When enabled, the NM will invoke callback ApplNwmVnActivationFailed() upon VN activation failures.

4.10 VNMF Message

The NM communicates with the different ECUs via the VNMF. The composition of the VNMF message is shown below:

VNMF messages always use an 11 bit CAN ID, defined by GM to be in the range 0x600 to 0x63F. VNMF messages contain 8 data bytes. The first data byte indicates the type of the message: If bit 0 is set, then the message is a VNMF-Init message. Otherwise, the message is a VNMF-Continue message. The remaining data bytes indicate which VN(s) are active.

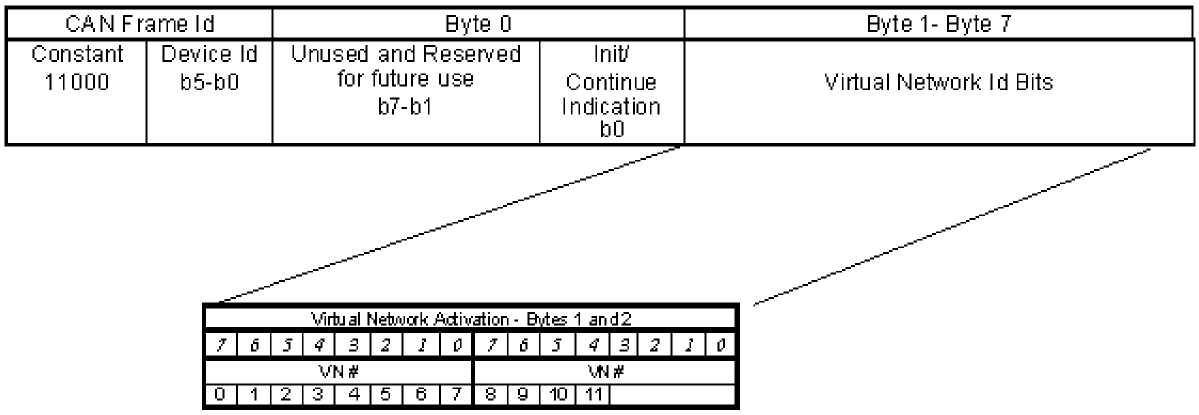


Figure 4-2 VNMF Message Layout

A VNMF-Init message is sent by NM whenever the application requests activation of one or more Network-Activated VNs. The initialization message will identify all active VNs managed by the ECU in addition to the new request(s).

Once the VNMF-Init message has been sent, NM will periodically send VNMF-Continue messages to keep the other ECUs informed of the active VNs.

Each data byte contains a bit-mask that identifies the active VN. VNs are assigned numeric values that are associated with a symbolic VN name in the message/signal database. The configuration tool generates a macro for each VN that the ECU participates in. The macros are defined in the file nm_cfg.h, and all have names of the form VN_<virtual network name>. The application should use these macros as the VN argument to all API functions that require a VN (e.g. IlNwmActivateVN`()`)

4.11 Fault Detection and Mitigation Algorithm

To enhance robustness of the NM, a fault detection and mitigation algorithm as specified in [2] observes the activation state of each VN and the global NM state of each channel.

There might be conditions that cause the NM to inadvertently keep single VNs or channels active, although all VNs and therefore the whole channel should not be active. This might happen e.g. due to single bit flips in NM internal variables. To prevent such situations and resulting battery drain situations, the algorithm performs consistency checks to detect inadvertent activation of VNs and NM channels.

Note
To save RAM, ROM and runtime the whole Fault Detection and Mitigation Algorithm can be disabled in GENy by the attribute 'Fault Detection and Mitigation Algorithm'. See chapter 6.3.2 'System-specific Configuration Options'.

In case a faulty activated VN or network has been detected the application will be notified by callback functions and the corresponding VN or network will be deactivated. In particular the following faults can be detected:

4.11.1 VN Active Fault

A 'VN Active Fault' is detected, if a VN that should be inactive is still active for more than twice the VN timer time (= 16s).

After the fault has been detected the algorithm will deactivate the corresponding VN and notify the application by calling ApplNwmVnActiveFault() (see also chapter 7.4 'Callback Functions').

4.11.2 Network Active Fault

A 'Network Active Fault' is detected, if the NM does not enter COMM-OFF state for more than twice the COMM-ENABLE timer time (= 16s) after the last VN has been deactivated.

After the fault has been detected the algorithm will start the shut-down sequence for the channel and the application is informed about the fault by the call of ApplNwmNetworkActiveFault() (see also chapter 7.4 'Callback Functions').

4.11.3 No Sleep Confirmation Fault

During shut-down the application has to optionally confirm the transition to sleep (ApplNwmSleepConfirmation()). If a shut-down is started due to the detection of a 'Network Active Fault', the algorithm is different:

A 'No Sleep Confirmation Fault' will be detected when the application does not confirm the transition to sleep for more than a configurable threshold value.

The concrete shut-down sequence depends on the following configuration attributes (See also chapter 6.3.2 'System-specific Configuration Options'):

> **Sleep Confirmation:** This attribute enables/disables the callback ApplNwmSleepConfirmation(). The callback informs the application that sleep mode can be entered and gives the application the control over sleep mode. Depending on the return value of the callback, sleep mode is directly entered or a time delay is triggered that results in another callback invocation after 8s.
> If this attribute is disabled, the Fault Detection Algorithm will directly shut down after 'Network Active Fault' has been detected, i.e. a 'No Sleep Confirmation Fault' will never be detected.

> **No Sleep Confirmation Fault Reporting:** If this attribute is enabled, the detection of a 'No Sleep Confirmation Fault' will be reported to application by calling ApplNwmNoSleepConfirmationFault (see also chapter 7.4 'Callback Functions').

> **No Sleep Confirmation Fault Mitigation:** If this attribute is enabled, the detection of a 'No Sleep Confirmation Fault' will cause the Fault Detection Algorithm to shut down the network even if the application still does not confirm sleep. If this attribute is disabled, the mitigation of 'No Sleep Confirmation Fault' has to be handled by application.

> **Max No Sleep Confirmation:** This value defines the threshold for the number of times the application may not confirm sleep before 'No Sleep Confirmation Fault' is detected. This attribute can be configured for each channel. See chapter 6.3.3 'Channel-specific Configuration Options'.

Note
The configuration options for the Fault Detection and Mitigation Algorithm might be not visible in GENy. In this case the attributes a pre-configured by Vector for your delivery.

5 Integration

5.1 Involved Files

To integrate the NM in your project, you need the following (static) embedded files:

File	Content
gmnm.c	Source file of the NM. Contains all API and algorithms.
gmnmdef.h	Header file of the NM. Contains all static prototypes for the API and definitions, such as symbolic constants.
_MemDef.h	Sample file for definitions of memory qualifies which will be used in gmlcal.c and gmlcal.h.

Table 5-1 Static Files

Additionally the following files have to be generated by the configuration tool (refer to chapter 6 'Configuration'):

File	Content
nm_cfg.h	Scales the NM and provides constants.
nm_par.c	Dynamic source file of the NM. Contains configuration tables.
nm_par.h	Dynamic header file of the NM.
gmlcal.c	Source file of the NM calibration data.
gmlcal.h	Header file of the NM calibration data

Table 5-2 Dynamic Files

5.2 Necessary Steps to Integrate the NM in Your Project

Following steps may be necessary to integrate the NM in your project:

> Copy the NM-related files into your project.

> Make these files available in your project settings, e.g. set the correct paths in your makefile.

> In order to make the NM available to your application, include the header file il_inc.h into all files that make use of NM services and functions.

> Start the configuration tool and configure the NM according to your needs (see chapter “6 Configuration”).

> Generate the configuration files (see chapter “6 Configuration”)

> Implement all necessary callbacks in your application (see chapter “7.4 Callback Functions”).

> Build your project (compile & link).

5.3 Necessary Steps to Run the NM

The NM should already have been integrated in your project and the building process should complete without any errors.

There are two main steps that have to be performed: Initialization and cyclic task calls.

Info
If you are using a CCL within the CANbedded stack, the initialization and the cyclic call of the task functions can be handled by the CCL. Please refer to the documentation of the CCL.

5.3.1 Initialization

The NM needs to be initialized only once after system start. Further calls for example after canceling of sleep mode are not necessary. To initialize the IL and the NM the function `IlInitPowerOn()` is provided by the IL. Please note that it is not allowed to call any function of these modules before they are initialized. Therefore the interrupts should be disabled until the module is initialized. If CAN driver will not be initialized by the NM, be sure that this is done before the first call of a cyclic IL or NM task function.

```
DisableAllInterrupts();
CanInitPowerOn(); /* if not handled by NM */
IlInitPowerOn(); /* initializes IL and NM */
ReenableAllInterrupts();
```

5.3.2 Periodic tasks

Add a cyclic function call of IlNwmTask() to your runtime environment. Ensure that the call cycle matches the value which is configured in the configuration tool.

The recommend order to call the NM and IL tasks is first NM and then IL:

```
IlNwmTask();
IlRxTask();
IlTxTask();
```

Caution
In preemptive systems and when CAN driver operates in polling mode it must be assured, that IlNwmTask does not interrupt NmConfirmation(), NmHVConfirmation(), NmPrecopy() and NmHVPrecopy().

5.4 Operating Systems

The CANbedded stack is designed and programmed to work with or without operating systems. Since the modules have to work without an operating system, resource locking mechanisms are not handled. To lock critical resources, interrupts will be disabled and restored. The CAN driver (Data Link Layer) provides functions to fulfill this task.

Each module has one or two functions (tasks) which have to be called periodically. For operating systems it is advisable to create one task and call all the NM module functions subsequently. To implement different periods of time, the OS task could have a counter to implement this.

To ensure data consistency on pre-emptive multi-tasking operating systems or when using kernel resources on interrupt level, there are two things to keep in mind.

> The kernel provides mechanisms to keep data consistent within multi-byte signals. That means, reading multi-byte data is always done while interrupts are locked. In that case, no task switch can occur. The disadvantage to that mechanism is a longer interrupt latency time. If your system is critical to long latency times, ensure that your system works properly in all cases.

> Bit field manipulation is done by macros. Some compilers and processors realize bit field manipulation by read-modify-write accesses. If data accesses to bit fields in the same byte are used on pre-emptive tasks or on interrupt level, a problem could be caused. Try to avoid this or make resource locking to such accesses.

5.5 Other Aspects

If the CAN controller is not capable to detect a CAN wakeup, the application must call API NmCanWakeUp() upon detection of a CAN wakeup event.

6 Configuration

6.1 Concept

The embedded component is configured with the help of a PC-based configuration tool named GENy. Settings for the NM can be selected in the GUI. These settings are used to generate the configuration files, which are needed to compile and run the component.

Some configuration options are based on information from the CAN database (DBC file). Some other options depend on the OEM and cannot be changed.

6.2 Data base attributes

The following table contains all attributes related to the Nm_Gmlan_Gm[1].

Attribute Name	Valid for	Type	Value	Description
BusOffRecoveryTime	Network	Integer	3500 (*)	This attribute defines the node recovery time after a BusOff event. This is an optional attribute.
BusWakeUpDelay	Network	Integer	100 (*)	This attribute defines the time between a High-Level Voltage Wakeup (HLVW) and the activation of Initially Active VNs. This parameter is also used as a delay time between the Activation of shared-local input VNs and the actual activation inside the ECU. This is an optional attribute.
NetworkType	Network	String	> Bodybus > Infotainment > Powertrain	This attribute defines the type of the network.
NmBaseAddress	Network	Hex	0x620	This attribute defines the base address of the NM messages (e.g. 0x620). Only a certain number of nodes can participate in the NM. The CAN identifiers of NM messages are kept in a certain range. This range starts with the ID given by attribute NmBaseAddress. The size of the range is given by attribute NmMessageCount.
NmMessage	Message	Enum	> no > yes (*)	This attribute defines if the corresponding message is a NM message (“Yes") or not (“No”). The CAN ID of this message must be within the range given by

[1] Default values are marked with (*).

				NmBaseAddress and NmMessageCount.
NmMessageCount	Network	Integer	32	This attribute defines the maximum number of nodes within the NM. The value is required to define the range precopy-function of the CAN driver's precopy function that is used to receive the NM messages. There is the requirement that the value of attribute NmMessageCount is 2^n (where n is a natural number).
NmNode	Node	Enum	> no > yes (*)	This attribute defines if the corresponding node uses the NM ("Yes") or not ("No").
NmType	Network	String	GMLAN (*)	This attribute defines the OEM-specific type of the NM. Must be set to "GMLAN".
NodeSuprvStabilityTime	Node	Integer	0..65535 5000 (*)	This attribute defines a delay time between activation of a VN and start of supervision of the corresponding signals. The default value is 5000ms. The supervision stability time is used to avoid 'Loss of Communication' DTCs due to transient conditions after VN activation.
SourceID	Node	Integer	0..255	The Source address of a node is given as an attribute inside the database. There are two possibilities to use this attribute: Instead of initialization of the Source Address by the application, the handler could do this in the function IlInit(). This would imply, that the value is always fixed (MIM-modules will be handled correctly depending on the pre-selection of the application, which instant should run). The transmit messages inside the handler will already be pre-set with the Source Address given in this attribute. This would avoid the runtime effort of the driver to add the source address every time a message must be transmitted. The dynamic setting will only be required for MIM's.
VN_<name>	Network	Integer	0..55	Bit number (0 to 55) within the VNMF message Network ID Bit field identifying the VN associated with the Network Name. Successive numbers must be given. "Holes" are not allowed.
VNECU<name>	node	Enum	> None > Activator > Remoted > Activator_Remoted > SharedLoc	This attribute defines the VN type for a specific ECU for the specific VN.

			al	
VNInitAct<name>	network	Enum	> no (*) > yes	This attribute defines whether a VN is declared as 'Initially Active'.
VNSig<name>	Signal	Enum	> no > yes (*)	If yes, indicates that the signal will be associated with the named virtual network. The name of the VN must match one of those defined for the network attribute VN_<name>,

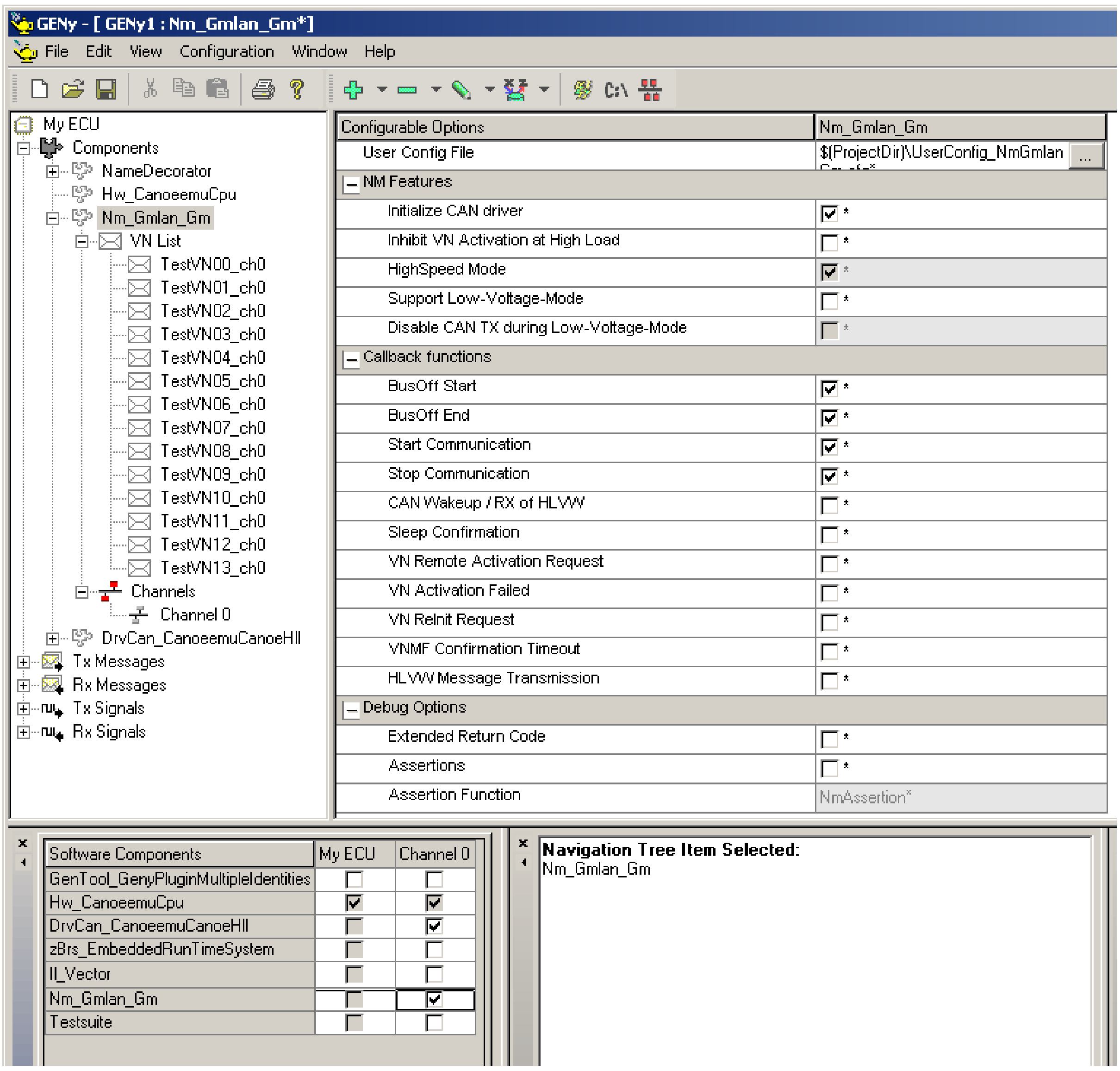
6.3 GENy

The configuration tool GENy is used to configure the CANbedded components. This tool generates source code and configuration files to make the CANbedded components run.

6.3.1 General

For a detailed description of the configuration tool and the description of the component-specific configuration options, please refer to the online-documentation within GENy.

	Info The screen shots in this chapter can differ from your screen because some options depend on the system setup.


GENy - [GENy1 : Nm_Gmlan_Gm*]
File Edit View Configuration Window Help

My ECU
- Components
  - NameDecorator
  - Hw_CanoeemuCpu
  - Nm_Gmlan_Gm
    - VN List
      - TestVN00_ch0
      - TestVN01_ch0
      - TestVN02_ch0
      - TestVN03_ch0
      - TestVN04_ch0
      - TestVN05_ch0
      - TestVN06_ch0
      - TestVN07_ch0
      - TestVN08_ch0
      - TestVN09_ch0
      - TestVN10_ch0
      - TestVN11_ch0
      - TestVN12_ch0
      - TestVN13_ch0
    - Channels
      - Channel 0
  - DrvCan_CanoeemuCanoeHll
- Tx Messages
- Rx Messages
- Tx Signals
- Rx Signals

Configurable Options	Nm_Gmlan_Gm
User Config File	$(ProjectDir)\UserConfig_NmGmlan
NM Features	
Initialize CAN driver	☑ *
Inhibit VN Activation at High Load	☐ *
HighSpeed Mode	☑ *
Support Low-Voltage-Mode	☐ *
Disable CAN TX during Low-Voltage-Mode	☐ *
Callback functions	
BusOff Start	☑ *
BusOff End	☑ *
Start Communication	☑ *
Stop Communication	☑ *
CAN Wakeup / RX of HLVW	☐ *
Sleep Confirmation	☐ *
VN Remote Activation Request	☐ *
VN Activation Failed	☐ *
VN ReInit Request	☐ *
VNMF Confirmation Timeout	☐ *
HLVW Message Transmission	☐ *
Debug Options	
Extended Return Code	☐ *
Assertions	☐ *
Assertion Function	NmAssertion*

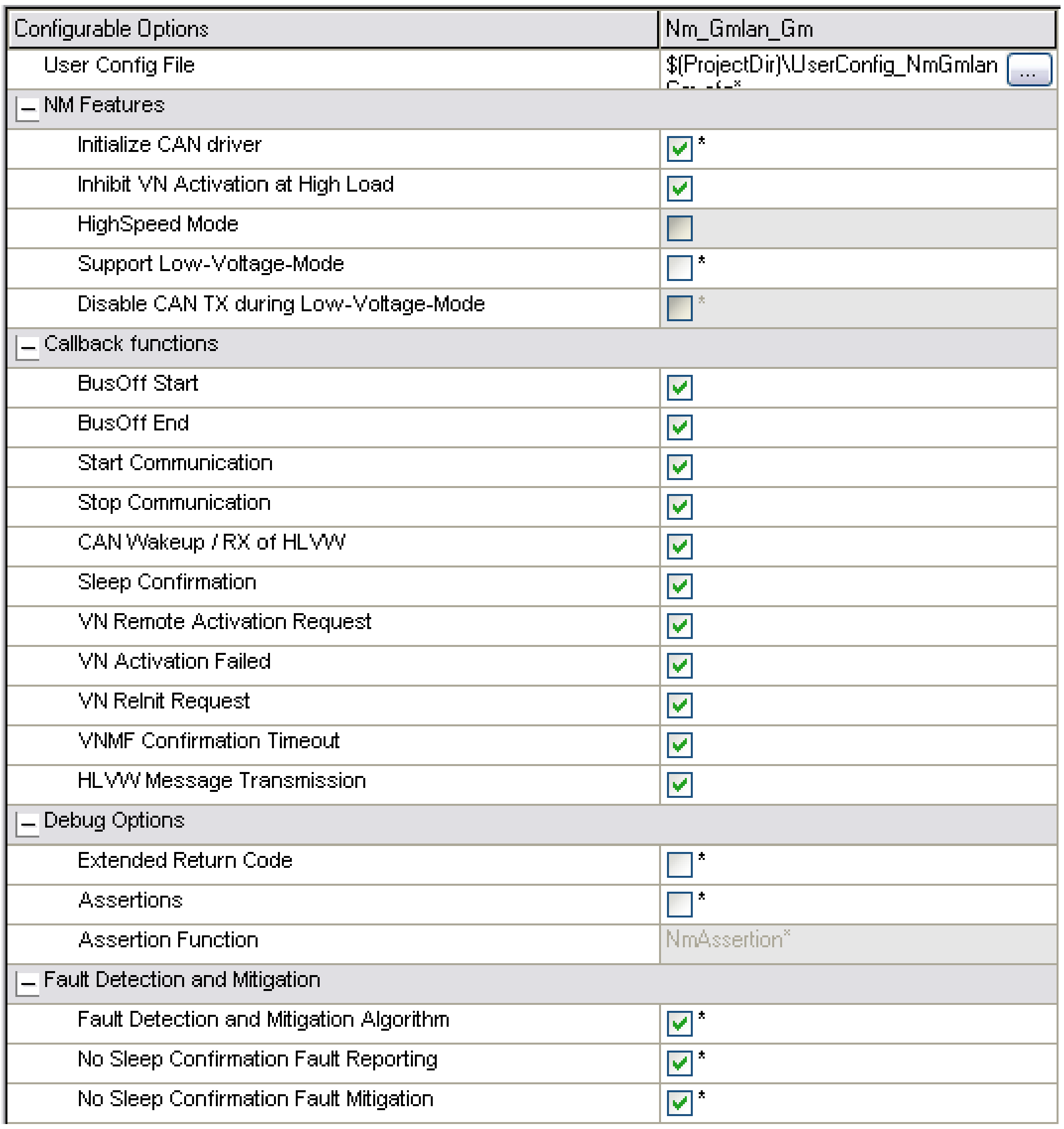
Software Components	My ECU	Channel 0
GenTool_GenyPluginMultipleIdentities	☐	☐
Hw_CanoeemuCpu	☑	☑
DrvCan_CanoeemuCanoeHll	☐	☑
zBrs_EmbeddedRunTimeSystem	☐	☐
Il_Vector	☐	☐
Nm_Gmlan_Gm	☐	☑
Testsuite	☐	☐

Navigation Tree Item Selected:
Nm_Gmlan_Gm

Figure 6-1 GENy Overview

6.3.2 System-specific Configuration Options

This configuration page allows to configure system-specific settings, e.g. the usage of available features of the component.

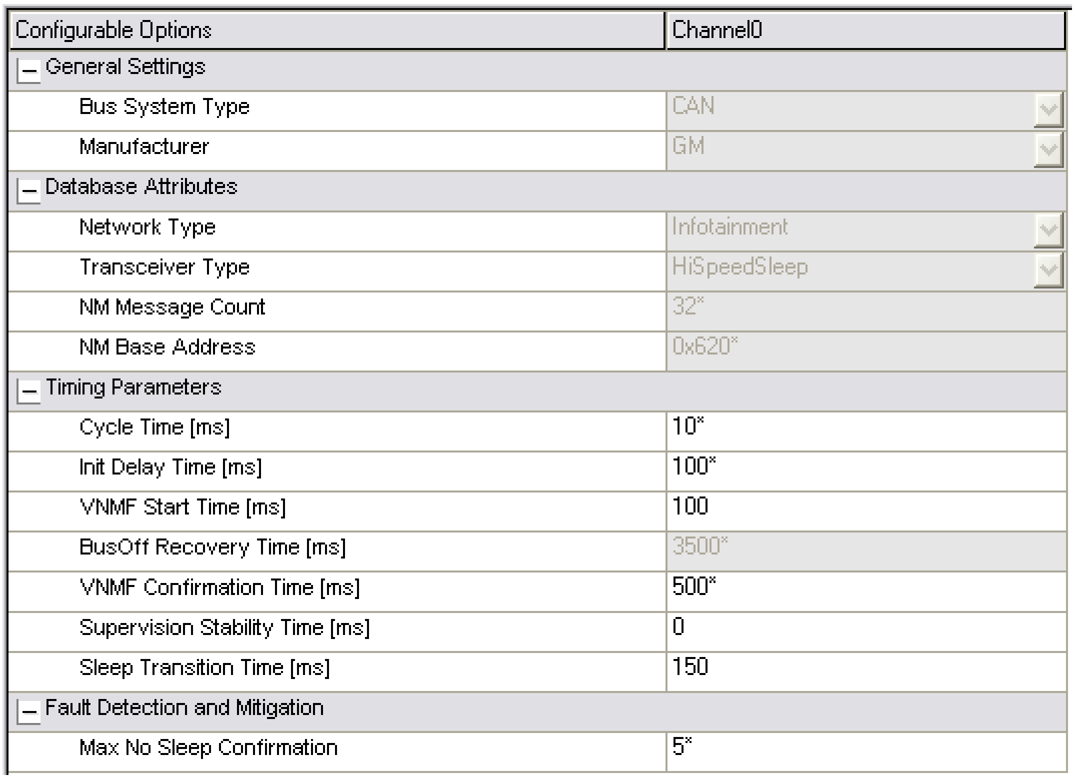


Configurable Options	Nm_Gmlan_Gm
User Config File	$(ProjectDir)\UserConfig_NmGmlan ...
NM Features	
Initialize CAN driver	✓ *
Inhibit VN Activation at High Load	✓
HighSpeed Mode	☐
Support Low-Voltage-Mode	☐ *
Disable CAN TX during Low-Voltage-Mode	☐ *
Callback functions	
BusOff Start	✓
BusOff End	✓
Start Communication	✓
Stop Communication	✓
CAN Wakeup / RX of HLVW	✓
Sleep Confirmation	✓
VN Remote Activation Request	✓
VN Activation Failed	✓
VN ReInit Request	✓
VNMF Confirmation Timeout	✓
HLVW Message Transmission	✓
Debug Options	
Extended Return Code	☐ *
Assertions	☐ *
Assertion Function	NmAssertion*
Fault Detection and Mitigation	
Fault Detection and Mitigation Algorithm	✓ *
No Sleep Confirmation Fault Reporting	✓ *
No Sleep Confirmation Fault Mitigation	✓ *

Figure 6-2 System-specific Configuration Options

6.3.3 Channel-specific Configuration Options

This configuration page allows to configure channel-specific settings, e.g. the network type and the timing for each channel.

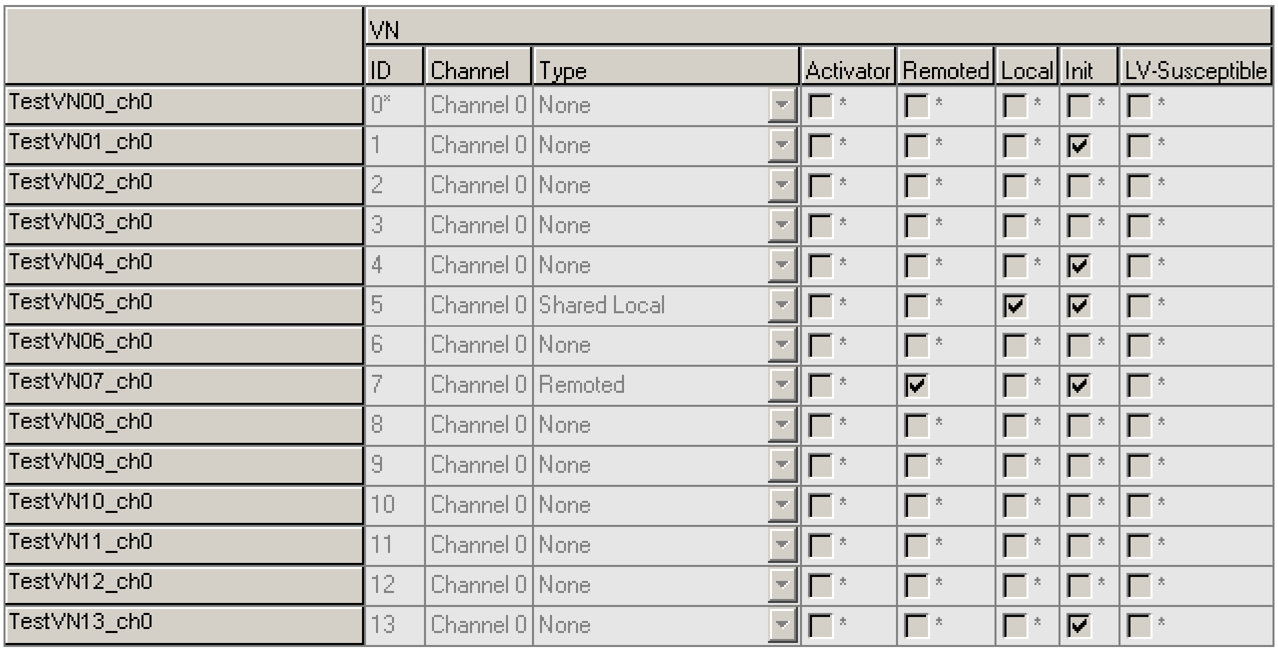


Configurable Options	Channel0
General Settings	
Bus System Type	CAN
Manufacturer	GM
Database Attributes	
Network Type	Infotainment
Transceiver Type	HiSpeedSleep
NM Message Count	32*
NM Base Address	0x620*
Timing Parameters	
Cycle Time [ms]	10*
Init Delay Time [ms]	100*
VNMF Start Time [ms]	100
BusOff Recovery Time [ms]	3500*
VNMF Confirmation Time [ms]	500*
Supervision Stability Time [ms]	0
Sleep Transition Time [ms]	150
Fault Detection and Mitigation	
Max No Sleep Confirmation	5*

Figure 6-3 Channel-specific Configuration Options

6.3.4 VN-specific Configuration Options

This configuration page gives an overview of the used VNs and allows to configure VN-specific settings, e.g. the LV-susceptible mode.



	VN							
	ID	Channel	Type	Activator	Remoted	Local	Init	LV-Susceptible
TestVN00_ch0	0*	Channel 0	None	☐ *	☐ *	☐ *	☐ *	☐ *
TestVN01_ch0	1	Channel 0	None	☐ *	☐ *	☐ *	☑	☐ *
TestVN02_ch0	2	Channel 0	None	☐ *	☐ *	☐ *	☐ *	☐ *
TestVN03_ch0	3	Channel 0	None	☐ *	☐ *	☐ *	☐ *	☐ *
TestVN04_ch0	4	Channel 0	None	☐ *	☐ *	☐ *	☑	☐ *
TestVN05_ch0	5	Channel 0	Shared Local	☐ *	☐ *	☑	☑	☐ *
TestVN06_ch0	6	Channel 0	None	☐ *	☐ *	☐ *	☐ *	☐ *
TestVN07_ch0	7	Channel 0	Remoted	☐ *	☑	☐ *	☑	☐ *
TestVN08_ch0	8	Channel 0	None	☐ *	☐ *	☐ *	☐ *	☐ *
TestVN09_ch0	9	Channel 0	None	☐ *	☐ *	☐ *	☐ *	☐ *
TestVN10_ch0	10	Channel 0	None	☐ *	☐ *	☐ *	☐ *	☐ *
TestVN11_ch0	11	Channel 0	None	☐ *	☐ *	☐ *	☐ *	☐ *
TestVN12_ch0	12	Channel 0	None	☐ *	☐ *	☐ *	☐ *	☐ *
TestVN13_ch0	13	Channel 0	None	☐ *	☐ *	☐ *	☑	☐ *

Figure 6-4 VN-specific Configuration Options

7 API Description

7.1 General

The NM uses different API prototypes. The usage depends on the number of channels in the system. Both prototypes differ in the usage of a channel parameter as the first function argument:

> The “Standard” prototype is used in single-channel applications. There is no channel parameter due to optimization.

> The “Indexed” prototype is used in multi-channel applications. The first argument is always the channel parameter.

7.2 Common Parameter

There are some parameters that are commonly used in multiple APIs.

Parameter	
channel	CAN channel handle
vnHndl	VN handle. Note: The VN handles are defined to symbolic names in nm_cfg.h. The symbolic name and the handle correspond to the value of the DBC network attribute named VN_<name>.

7.3 Service Functions

NM Handling

> IlNwmTask

NM status

> IlNwmGetStatus

LVT mode

> IlNwmEnterLowVoltageMode
> IlNwmExitLowVoltageMode

Diagnostic mode

> IlNwmNormalCommHalted
> IlNwmReturnToNormalMode

HighSpeed mode

> IlNwmSetHispeedMode
> IlNwmResetHispeedMode

VN activation

> IlNwmActivateVN
> IlNwmDeactivateVN
> IlNwmAllowActivationVN
> IlNwmInhibitActivationVN

VN status

> IlNwmGetActiveListVN
> IlNwmIsActiveVN

Context Switch

> NmGetModuleContext
> NmSetModuleContext

IlNwmActivateVN

Prototype	
Standard	`Nm_Status **IlNwmActivateVN**( vuint8 VnHndl )`
Indexed	`Nm_Status **IlNwmActivateVN** ( CanChannelHandle channel, vuint8 VnHndl )`
Parameter	
	see “7.2 Common Parameter”
Return code	
Nm_Status	> NM_OK The activation request was accepted. > NM_ACTIVE The VN is already active > NM_ERROR The node is in the HighSpeed mode. > NM_HALTED The node is in the NormalCommHalted mode. > NM_INACTIVE The application is not permitted to activate the VN (The node is neither an Activator nor Local.) > NM_WRONG_ARG VnHndl is invalid
Functional Description	
This function activates a VN given by VN handle <VnHndl>. This function may only be called for VNs that are meant to be activated (Activator, Local). When activation is complete, the callback function ApplNwmVnActivated() is executed. Related API: IlNwmDeactivateVN()	
Particularities and Limitations	
>	

IlNwmAllowActivationVN

Prototype	
Standard	`void **IlNwmAllowActivationVN** ( void )`
Indexed	`void **IlNwmAllowActivationVN** ( CanChannelHandle channel )`
Parameter	
	see “7.2 Common Parameter”
Return code	

Functional Description	
This function restores the VN activation. The NM will start all queued VN activation requests. Related API: IlNwmInhibitActivationVN()	
Particularities and Limitations	
> Availability of this function can be configured in GENy (“Inhibit VN Activation at Highload”). > Only available if there are Activator VNs.	

IlNwmDeactivateVN

Prototype	
Standard	`Nm_Status **IlNwmDeactivateVN**( vuint8 VnHndl )`
Indexed	`Nm_Status **IlNwmDeactivateVN** ( CanChannelHandle channel, vuint8 VnHndl )`
Parameter	
	see “7.2 Common Parameter”
Return code	
Nm_Status	> NM_OK The deactivation request was accepted. > NM_INACTIVE The VN was already deactivated. > NM_WRONG_ARG VnHndl is invalid.
Functional Description	
This function de-activates a VN given by VN handle <VnHndl>. Local VNs are immediately de-activated. Activator VNs are deactivated when the VN-specific timeout timer expires. The NM starts a VN timer for this VN. May only be called for VNs that are meant to be activated (Activator, Local). When de-activation is complete, the callback function ApplNwmVnDeactivated() is executed. Related API: IlNwmActivateVN()	
Particularities and Limitations	
>	

IlNwmEnterLowVoltageMode

Prototype	
Standard	`void **IlNwmEnterLowVoltageMode**( void )`
Indexed	`void **IlNwmEnterLowVoltageMode**( CanChannelHandle channel )`
Parameter	
	see “7.2 Common Parameter”
Return code	

Functional Description	
This function activates the Low Voltage Tolerant (LVT) Mode. The transmission of HLVW is disabled. VN monitoring and signal supervision timers are disabled. Please refer to “4.3 Low Voltage Tolerant Mode” for details. Related API: IlNwmExitLowVoltageMode()	
Particularities and Limitations	
> This function is only available if the LVT mode is enabled.	

IlNwmExitLowVoltageMode

Prototype	
Standard	`void IlNwmExitLowVoltageMode( void )`
Indexed	`void IlNwmExitLowVoltageMode( CanChannelHandle channel )`
Parameter	
	see “7.2 Common Parameter”
Return code	

Functional Description	
This function de-activates the Low Voltage Tolerant (LVT) Mode. VN monitoring and signal supervision timers are re-enabled. VNs that are LV-susceptible will be initialized and reactivated. Please refer to “4.3 Low Voltage Tolerant Mode” for details. Related API: IlNwmEnterLowVoltageMode()	
Particularities and Limitations	
> This function is only available if the LVT mode is enabled.	

IlNwmGetActiveListVN

Prototype	
Standard	`void IlNwmGetActiveListVN (vuint8 *VnList)`
Indexed	`void IlNwmGetActiveListVN ( CanChannelHandle channel, vuint8 *VnList)`
Parameter	
*VnList	pointer to an application-defined array that should contain the VN activation status. The application should declare the array as: vuint8 VnList [(<VNs>+7)/8] VNs = Amount of used VNs in this CAN channel (Remote, Activator and Local VNs). see “7.2 Common Parameter”
Return code	

Functional Description

This function retrieves the activation state of all VNs on the current channel.

The array pointed to by VnList is filled with a bit-mask. Each bit represents the status of a VN. If a bit is set, the corresponding VN is active. Otherwise not.

The most significant bit of the first byte represents VN 0. For example:

Byte 0								Byte 1								Byte
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	Bit
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	VN

Particularities and Limitations

>

IlNwmGetStatus

Prototype	
Standard	`nmStatusType` **`IlNwmGetStatus`** `( void )`
Indexed	`nmStatusType` **`IlNwmGetStatus`** `( CanChannelHandle channel )`
Parameter	
	see “7.2 Common Parameter”
Return code	
`nmStatusType`	Flag field that contains the network status.
Functional Description	

This function returns information about the NM status. The status is provided by a flag field within the return value. This return value can be decoded with the macros shown below. Each macro will return true or false. True indicates that the NM is within the specified mode:

IlNwmStateNormalCommHalted (status)	NormalCommHalted mode
INwmStateHispeedMode (status)	HighSpeed mode
IlNwmStateNoCommunication (status)	Network communications are disabled. No messages can be received or transmitted (BusOff or COMM-OFF state)
IlNwmStateSleepModeEntered (status)	Sleep mode (COMM-OFF state)
IlNwmStateSleepModePending (status)	Sleep mode is pending (COMM-ENABLE state)
IlNwmStateBusOffOccured (status)	At least 1 BusOff has occurred since the last activation of the node.
IlNwmStateNMActive (status)	NM is active. At least 1 associated VN is active (COMM-ACTIVE state).
IlNwmStateLowVoltageMode (status)	LVT mode is active

Particularities and Limitations

>

IlNwmInhibitActivationVN

Prototype	
Standard	`void` **`IlNwmInhibitActivationVN`** `( void )`
Indexed	`void` **`IlNwmInhibitActivationVN`** `( CanChannelHandle channel )`
Parameter	
	see “7.2 Common Parameter”
Return code	

Functional Description	

VN activation is suspended. All requests to start a VN are queued until VN activation is allowed.

Related API: IlNwmAllowActivationVN().

Particularities and Limitations

> Only available if enabled in the configuration

IlNwmIsActiveVN

Prototype	
Standard	`vuint8 IlNwmIsActiveVN( vuint8 VnHndl )`
Indexed	`vuint8 IlNwmIsActiveVN( CanChannelHandle channel, vuint8 VnHndl )`
Parameter	
	see "7.2 Common Parameter"
Return code	
	Status of the queried VN; Any non-zero value indicates that the VN is active.
Functional Description	
This function returns a flag field that contains the current state of a specific VN (given by VN handle <VnHndl>). If the return value is 0, the VN is inactive, otherwise active. The NM provides function macros that can evaluate the return value to get more information on the VN state. The macros evaluate to be true or false. IlNwmIsNmVnActivatorPending(vnState) Application has requested activation of VN IlNwmIsNmVnActive(vnState) VN is completely activated IlNwmIsNmVnActivator(vnState) Send out VNMF for this VN IlNwmIsNmVnLocal(vnState) VN is locally active IlNwmIsNmVnRxActive(vnState) Indicates Receive-enabled IlNwmIsNmVnNone(vnState) VN is not active	
Particularities and Limitations	
>	

IlNwmNormalCommHalted

Prototype	
Standard	`Nm_Status IlNwmNormalCommHalted( void )`
Indexed	`Nm_Status IlNwmNormalCommHalted( CanChannelHandle channel )`
Parameter	
	see "7.2 Common Parameter"
Return code	
	> NM_OK Node state was changed to Normal Comm. Halted. > NM_ERROR Error if the node is in the HighSpeed, Sleep, or LVT mode
Functional Description	
This function initiates diagnostic mode communications. All active VNs are deactivated and the diagnostic VN (VN 0) is activated. Note: If the node was in HighSpeed mode before this call, the CAN controller and transceiver will be reset into Normal mode by this function. Also refer to chapter "4.6 Normal Communication Halted" Related API: IlNwmReturnToNormalMode()	

Particularities and Limitations
>

IlNwmResetHispeedMode

Prototype	
Standard	`void IlNwmResetHispeedMode( void )`
Indexed	`void IlNwmResetHispeedMode( CanChannelHandle channel )`
Parameter	
	see “7.2 Common Parameter”
Return code	

Functional Description	
This function puts the transceiver and CAN controller in normal (low-speed) mode. The diagnostics VN (VN 0) is deactivated. After reinitializing the CAN controller, the user-defined function ApplTrcvrNormalMode() will be called, as the application is responsible for switching the transceiver to normal mode. After a delay, Rx and Tx functions of the IL are restarted. Related API: IlNwmSetHispeedMode()	
Particularities and Limitations	
>	

IlNwmReturnToNormalMode

Prototype	
Standard	`void IlNwmReturnToNormalMode( void )`
Indexed	`void IlNwmReturnToNormalMode( CanChannelHandle channel )`
Parameter	
	see “7.2 Common Parameter”
Return code	

Functional Description	
This function requests the NM to return to the normal communication mode. VNs cannot be activated during NormalCommHalted mode. Note: If the node was in HighSpeed mode before this call, the CAN controller and transceiver will be reset into normal mode by this function. Also refer to chapter “4.6 Normal Communication Halted” Related API: IlNwmNormalCommHalted()	
Particularities and Limitations	
>	

IlNwmSetHispeedMode

Prototype	
Standard	`Nm_Status **IlNwmSetHispeedMode**( void )`
Indexed	`Nm_Status **IlNwmSetHispeedMode**( CanChannelHandle channel )`
Parameter	
	see "7.2 Common Parameter"
Return code	
	> NM_OK — Node state was changed to HighSpeed. > NM_ERROR — Error if the node is in the NormalCommHalted or Sleep state
Functional Description	
This function requests the NM to switch over to HighSpeed mode. The transceiver and CAN controller are set to a higher data transmission rate. After switching to HighSpeed mode, the user-defined function ApplTrcvrHighSpeed() is called (the application is responsible for switching the transceiver into high-speed mode). Before calling this function, the application should place the GMLAN handler into the NormalCommHalted mode. Note that this function is only available for devices configured for single-wire CAN (Bodybus). This feature is only available if enabled in the configuration. Related API: IlNwmResetHispeedMode()	
Particularities and Limitations	
>	

IlNwmTask

Prototype	
Standard	`void **IlNwmTask**( void )`
Indexed	`void **IlNwmTask**( CanChannelHandle channel )`
Parameter	
	see "7.2 Common Parameter"
Return code	

Functional Description	
This function is the NM cyclic task function. The function is responsible for VN activation/deactivation, reception/transmission of HLVW messages, mode and state handling. The user-application is responsible for periodically calling this function at a user-defined timing. This timing can be configured in GENy.	
Particularities and Limitations	
> In preemptive systems and when CAN driver operates in polling mode it must be assured, that IlNwmTask does not interrupt NmConfirmation(), NmHVConfirmation(), NmPrecopy() and NmHVPrecopy().	

NmGetModuleContext

Prototype	
`void NmGetModuleContext( tNmModuleContextStructPtr pContext )`	
Parameter	
`pContext`	Pointer to structure which is filled by this function with the current module context. The type definition is provided in the modules header file.
Return code	
–	-
Functional Description	
This function copies all internal and global variables of the network management into the passed structure.	
Particularities and Limitations	
> The function is usually called by the QNX Mini driver wrapper. > The function is only available if the QNX Mini driver wrapper is enabled or NM_ENABLE_GET_CONTEXT is defined in a user configuration file.	
Call context	
> Task level only	

NmSetModuleContext

Prototype	
`vuint8 NmSetModuleContext( tNmModuleContextStructPtr pContext )`	
Parameter	
`pContext`	Pointer to structure which is filled by this function with the current module context. The type definition is provided in the modules header file.
Return code	
	1: The magic number member of the passed structure matches the version of the module. 0: The magic number does not match. The data from the passed structure is not copied.
Functional Description	
This function initializes all internal and global variables of the network management with the values from the passed structure.	
Particularities and Limitations	
> The function is usually called by the QWrapper module > The function is only available if the QWrapper module is enabled or NM_ENABLE_SET_CONTEXT is defined in a user configuration file.	
Call context	
> Task level only	

7.4 Callback Functions

Transceiver Handling

> ApplTrcvrHighSpeed
> ApplTrcvrHighVoltage
> ApplTrcvrNormalMode
> ApplTrcvrSleepMode

VN Handling

> ApplNwmReinitRequest
> ApplNwmVnActivated
> ApplNwmVnActivationFailed
> ApplNwmVnDeactivated
> ApplNwmVnmfConfirmationTimeout
> ApplNwmVnRemoteActivateRequest

Wakeup/Sleep Handling

> ApplNwm100MsgRecv
> ApplNwmHLVWStart
> ApplNwmHLVWStop
> ApplNwmSleep
> ApplNwmSleepConfirmation
> ApplNwmWakeup
> ApplNwmWakeupMsgRecv

BusOff Handling

> ApplNwmBusoff
> ApplNwmBusoffEnd

Fault Detection & Mitigation Algorithm

> ApplNwmVnActiveFault
> ApplNwmNetworkActiveFault
> ApplNwmNoSleepConfirmationFault

ApplNwm100MsgRecv

Prototype	
Standard	`void ApplNwm100MsgRecv ( void )`
Indexed	`void ApplNwm100MsgRecv ( CanChannelHandle channel )`
Parameter	
	see “7.2 Common Parameter”
Return code	

Functional Description	
This callback is executed to notify the application that a HLVW message was received with the CAN controller being in an active state.	
Particularities and Limitations	
> The availability of this callback can be configured: NM_ENABLE_WAKEUP_RECEIVED_FCT	
Call context	
> This function is called from task level only.	

ApplNwmBusoff

Prototype	
Standard	`void ApplNwmBusoff ( void )`
Indexed	`void ApplNwmBusoff ( CanChannelHandle channel )`
Parameter	
	see “7.2 Common Parameter”
Return code	

Functional Description	
This callback is executed to notify the application that the CAN controller has entered BusOff state, indicating that errors have occurred on the CAN bus. Transmission and reception of messages are disabled at this time. The NM starts a recovery timer. The recovery time can be selected in the configuration tool. When the recovery time elapses, the NM will re-enable the CAN controller. Related API: ApplNwmBusoffEnd()	
Particularities and Limitations	
> The availability of this callback can be configured: NM_ENABLE_BUSOFF_FCT > This callback might be executed in the CAN driver’s ISR. The application should exit this function as quickly as possible, and should not call any other NM or CAN API functions.	
Call context	
> Same as CAN driver error handling	

ApplNwmBusoffEnd

Prototype	
Standard	`void ApplNwmBusoffEnd ( void )`
Indexed	`void ApplNwmBusoffEnd ( CanChannelHandle channel )`
Parameter	
	see “7.2 Common Parameter”
Return code	

Functional Description	
This callback is executed to notify the application that the CAN controller has recovered from a BusOff state. Transmission and reception of messages are re-enabled. Related API: ApplNwmBusoff()	
Particularities and Limitations	
> The availability of this callback can be configured: NM_ENABLE_BUSOFF_END_FCT	
Call context	
> This function is called from task level only.	

ApplNwmHLVWStart

Prototype	
Standard	`void ApplNwmHLVWStart ( void )`
Indexed	`void ApplNwmHLVWStart ( CanChannelHandle channel )`
Parameter	
	see “7.2 Common Parameter”
Return code	

Functional Description	
This callback is executed just before the transmission of a HLVW message. Related API: ApplNwmHLVWStop()	
Particularities and Limitations	
> The availability of this callback can be configured: NM_ENABLE_HLVW_INDICATION_FCT	
Call context	
> If CAN driver transmit queue is activated this function may be called in interrupt context.	

ApplNwmHLVWStop

Prototype	
Standard	`void **ApplNwmHLVWStop** ( void )`
Indexed	`void **ApplNwmHLVWStop** ( CanChannelHandle channel )`
Parameter	
	see “7.2 Common Parameter”
Return code	

Functional Description	
This callback is executed just after the transmission of a HLVW message has completed (and confirmed). Related API: ApplNwmHLVWStart()	
Particularities and Limitations	
> The availability of this callback can be configured: NM_ENABLE_HLVW_INDICATION_FCT	
Call context	
> This function will maybe called in interrupt context.	

ApplNwmReinitRequest

Prototype	
Standard	`void ApplNwmReinitRequest ( vuint8 VnHndl, vuint8 ReinitRequest )`
Indexed	`void ApplNwmReinitRequest ( CanChannelHandle channel, vuint8 VnHndl, vuint8 ReinitRequest )`
Parameter	
ReinitRequest	Reason for Re-init request > 0 A VNMF-Continue message was received for an inactive VN. > 1 A VNMF-Init message was received for an already active VN. > 2 A VNMF-Init message was transmitted for an already active VN. This is the case when an Activator VN is (re-)activated and VN is still active and VN timer is less than 4 seconds. see “7.2 Common Parameter”
Return code	

Functional Description	
This callback is executed to notify the application that a VN is being re-initialized. The reason for the re-initialization is given as parameter.	
Particularities and Limitations	
> The availability of this callback can be configured: NM_ENABLE_REINITREQUEST_FCT	
Call context	
> This function is called from task level only.	

ApplNwmSleep

Prototype	
Standard	`void ApplNwmSleep ( void )`
Indexed	`void ApplNwmSleep ( CanChannelHandle channel )`
Parameter	
	see “7.2 Common Parameter”
Return code	
Functional Description	
This callback is executed to notify the application that NM is entering the COMM-OFF state. The sleep indication may be used by the application to suspend network (and other) operations, including cyclic calls to the NM task functions. Related API: ApplNwmWakeup()	
Particularities and Limitations	
> The availability of this callback can be configured: NM_ENABLE_SLEEP_FCT	
Call context	
> This function is called from task level only.	

ApplNwmSleepConfirmation

Prototype	
Standard	`vuint8 ApplNwmSleepConfirmation ( void )`
Indexed	`vuint8 ApplNwmSleepConfirmation ( CanChannelHandle channel )`
Parameter	
	see “7.2 Common Parameter”
Return code	
	> NmSleepOk Enter Sleep mode > NmSleepNo Stay awake for another 8 seconds.
Functional Description	
The callback is executed to let the application decide if the NM can enter Sleep Mode. If the application returns NmSleepNo from the callback, then the network will stay enabled for another 8 seconds. Unless other events intervene (such as VN activation), NM will call this function again when the timer expires.	
Particularities and Limitations	
> The availability of this callback can be configured: NM_ENABLE_SLEEPCONFIRMATION_FCT	
Call context	
> This function is called from task level only.	

ApplNwmVnActivationFailed

Prototype	
Standard	`vuint8 ApplNwmVnActivationFailed ( vuint8 VnHndl )`
Indexed	`vuint8 ApplNwmVnActivationFailed ( CanChannelHandle channel, vuint8 VnHndl )`
Parameter	
	see “7.2 Common Parameter”
Return code	
	> 0 Deactivate the VN > 1 Reattempt activation of the VN.
Functional Description	
This callback is executed to notify the application that an attempt to activate a VN has failed. The failure is detected when the VN active timer counts down to 1 second while an activation attempt still pending. This function gives the application the ability to decide whether to retry or abort the VN activation. The return value from the function (see above) controls the behavior of NM. If this callback is not enabled, then NM will deactivate the VN and does not reattempt the activation.	
Particularities and Limitations	
> The availability of this callback can be configured: NM_ENABLE_VN_ACTIVATION_FAILED_FCT	
Call context	
> This function is called from task level only.	

ApplNwmVnActivated

Prototype	
Standard	`void ApplNwmVnActivated (vuint8 VnHndl )`
Indexed	`void ApplNwmVnActivated ( CanChannelHandle channel, vuint8 VnHndl )`
Parameter	
	see “7.2 Common Parameter”
Return code	

Functional Description	
The callback is executed to notify the application that a VN has been activated. Related API: ApplNwmVnDeactivated()	
Particularities and Limitations	
> Mandatory callback	
Call context	
> This function is called from task level only.	

ApplNwmVnDeactivated

Prototype	
Standard	`void` **`ApplNwmVnDeactivated`** `(vuint8 VnHndl )`
Indexed	`void` **`ApplNwmVnDeactivated`** `( CanChannelHandle channel, vuint8 VnHndl )`
Parameter	
	see “7.2 Common Parameter”
Return code	
Functional Description	
The callback is executed to notify the application that a VN has been deactivated. Related API: ApplNwmVnActivated()	
Particularities and Limitations	
> Mandatory callback	
Call context	
> This function is called from task level only.	

ApplNwmVnRemoteActivateRequest

Prototype	
Standard	`vuint8` **`ApplNwmVnRemoteActivateRequest`** `( vuint8 VnHndl )`
Indexed	`vuint8` **`ApplNwmVnRemoteActivateRequest`** `( CanChannelHandle channel, vuint8 VnHndl )`
Parameter	
	see “7.2 Common Parameter”
Return code	
	The application must return a value indicating if the activation request should be accepted or rejected. > 1 Accept remote activation. > 0 Reject remote activation.
Functional Description	
This callback is executed when a VN activation request (VNMF-Init) message is received. This allows the application to be notified of the activation, and to allow the activation request to be denied. Note: It is not recommended that the application reject activation requests. If this callback is disabled, the NM will accept the activation request.	
Particularities and Limitations	
> The availability of this callback can be configured: NM_ENABLE_VN_REMOTE_ACTIVATE_REQUEST_FCT	
Call context	
> This function is called from task level only.	

ApplNwmVnmfConfirmationTimeout

Prototype	
Standard	`void` **`ApplNwmVnmfConfirmationTimeout`** `( void )`
Indexed	`void` **`ApplNwmVnmfConfirmationTimeout`** `( CanChannelHandle channel )`
Parameter	
	see “7.2 Common Parameter”
Return code	

Functional Description	
This callback is executed to notify the application that transmission of a VNMF could not be completed within the configured time limit.	
Particularities and Limitations	
> The availability of this callback can be configured: NM_ENABLE_VNMF_CONFIRMATION_TIMEOUT_FCT	
Call context	
> This function is called from task level only.	

ApplNwmWakeup

Prototype	
Standard	`void` **`ApplNwmWakeup`** `( void )`
Indexed	`void` **`ApplNwmWakeup`** `( CanChannelHandle channel )`
Parameter	
	see “7.2 Common Parameter”
Return code	

Functional Description	
This callback is executed to notify the application that the NM is leaving the COMM-OFF state, and entering either COMM-ENABLED or COMM-ACTIVE. Note: If the application does not call the cyclic task of the NM while the CAN is in sleep mode, the callback ApplNwmWakeupMsgRecv() must be activated. If it is called, application has to re-enable the cyclic call. Related API: ApplNwmSleep()	
Particularities and Limitations	
> The availability of this callback can be configured: NM_ENABLE_WAKEUP_FCT	
Call context	
> This function is called from task level only.	

ApplNwmWakeupMsgRecv

Prototype	
Standard	`void` **`ApplNwmWakeupMsgRecv`** `( void )`
Indexed	`void` **`ApplNwmWakeupMsgRecv`** `( CanChannelHandle channel )`
Parameter	
	see “7.2 Common Parameter”
Return code	

Functional Description	
This callback is executed to notify the application that a HLVW message was received when the CAN controller was asleep. This may be used by the application to restart network activities, such as invoking the NM task functions periodically, or prevent entering the STOP mode.	
Particularities and Limitations	
> The availability of this callback can be configured: NM_ENABLE_WAKEUP_RECEIVED_FCT > This callback might be executed in the CAN driver’s ISR. The application should exit this function as quickly as possible, and should not call any other NM or CAN API functions.	
Call context	
> Same as CAN driver wakeup handling.	

ApplTrcvrHighSpeed

Prototype	
Standard	`void` **`ApplTrcvrHighSpeed`** `( void )`
Indexed	`void` **`ApplTrcvrHighSpeed`** `( CanChannelHandle channel )`
Parameter	
	see “7.2 Common Parameter”
Return code	

Functional Description	
This callback is executed when the transceiver has to be set to HighSpeed mode. The application has to set the transceiver in the corresponding state.	
Particularities and Limitations	
> This callback is available if a single-wire transceiver is used and HighSpeed mode is enabled	
Call context	
> Same as IlNwmSetHispeedMode.	

ApplTrcvrHighVoltage

Prototype	
Standard	`void ApplTrcvrHighVoltage( void )`
Indexed	`void ApplTrcvrHighVoltage( CanChannelHandle channel )`
Parameter	
	see “7.2 Common Parameter”
Return code	

Functional Description	
This callback is executed when a HLVW message has to be transmitted. The application has to set the transceiver in the corresponding state where High-Voltage transmission is possible.	
Particularities and Limitations	
> This callback is available if a single-wire transceiver is used	
Call context	
> If CAN driver transmit queue is activated this function will maybe called in interrupt context.	

ApplTrcvrNormalMode

Prototype	
Standard	`void ApplTrcvrNormalMode( void )`
Indexed	`void ApplTrcvrNormalMode( CanChannelHandle channel )`
Parameter	
	see “7.2 Common Parameter”
Return code	
Functional Description	
This callback is executed when the transceiver has to be set to normal mode, e.g. after transmission of a HLVW message. The application has to set the transceiver in the corresponding state (normal mode).	
Particularities and Limitations	
> Mandatory callback: always used	
Call context	
> This function may be called in interrupt context.	

ApplTrcvrSleepMode

Prototype	
Standard	`void` **`ApplTrcvrSleepMode`** `( void )`
Indexed	`void` **`ApplTrcvrSleepMode`** `( CanChannelHandle channel )`
Parameter	
	see “7.2 Common Parameter”
Return code	

Functional Description	
This callback is executed when the transceiver has to be set to sleep mode, i.e. the NM enters COMM-OFF state. The application has to set the transceiver in the corresponding state (sleep mode).	
Particularities and Limitations	
> This callback is available if a sleep/wake-able transceiver is used (single-wire, HighSpeed with sleep)	
Call context	
> This function is called from task level only.	

ApplNwmVnActiveFault

Prototype	
Standard	`void ApplNwmVnActiveFault ( vuint8 vnHndl )`
Indexed	`void ApplNwmVnActiveFault ( CanChannelHandle channel, vuint8 vnHndl )`
Parameter	
	see “7.2 Common Parameter”
Return code	

Functional Description	
This callback informs the application about a ‘VN Active Fault’ detected by the ‘Fault Detection and Mitigation Algorithm’. Please refer to chapter 4.11 for more information.	
Particularities and Limitations	
> This callback is available only if ‘Fault Detection and Mitigation Algorithm’ is enabled in GENy.	
Call context	
> This function is called from task level only.	

ApplNwmNetworkActiveFault

Prototype	
Standard	`void ApplNwmNetworkActiveFault ( void )`
Indexed	`void ApplNwmNetworkActiveFault ( CanChannelHandle channel )`
Parameter	
	see “7.2 Common Parameter”
Return code	

Functional Description	
This callback informs the application about a ‘Network Active Fault’ detected by the ‘Fault Detection and Mitigation Algorithm’. Please refer to chapter 4.11 for more information.	
Particularities and Limitations	
> This callback is only available if ‘Fault Detection and Mitigation Algorithm’ is enabled in GENy.	
Call context	
> This function is called from task level only.	

ApplNwmNoSleepConfirmationFault

Prototype	
Standard	`void` **`ApplNwmNoSleepConfirmationFault`** `( void )`
Indexed	`void` **`ApplNwmNoSleepConfirmationFault`** `( CanChannelHandle channel )`
Parameter	
	see “7.2 Common Parameter”
Return code	

Functional Description	
This callback informs the application about a ‘No Sleep Confirmation Fault’ detected by the ‘Fault Detection and Mitigation Algorithm’. Please refer to chapter 4.11 for more information.	
Particularities and Limitations	
> This callback is available only if ‘Fault Detection and Mitigation Algorithm’ and ‘No Sleep Confirmation Fault Reporting’ is enabled in GENy.	
Call context	
> This function is called from task level only.	

7.5 Calibration Constants

Please see TechnicalReferenceGMLANCalibration.pdf for more details on calibrate constants of the GMLAN handler.

8 Glossary and Abbreviations

8.1 Glossary

Term	Description
CAN Driver	The CAN Driver represents an implementation of the Data Link Layer by Vector Informatik GmbH.
CANbedded	CANbedded stands for a group of products offered by Vector Informatik GmbH including communication modules for CAN communication and a configuration tool to configure these modules.
Communication Database	See Network Database
Data Dictionary	See Network Database
Data Link Layer	The Data Link Layer defines the connection between two network nodes in the same network. It is responsible for error detection, error correction and flow control.
Deadline Monitoring	Deadline Monitoring is used to monitor the receipt of periodic messages related to the ECU. Each time a periodic message is received a timer or counter will be restarted. If the timer elapses the application will be notified.
Delay Time	To prevent high bus load the Interaction Layer waits a defined time after the transmission of a message before transmitting the next message. A delay time is related to a specific message.
Configuration Tool	Tool to generate parts of the code of the communication modules. The generated code will be specific for an application and will include the interface for the signals, messages, flags ...
Interaction Layer	By the Interaction Layer the data is structured. It is responsible for the consistency of the data offered to the upper layer.
Message	Variable for data exchange of which the length depends on the length of the frames used by the underlying field bus. CAN for example use 0-8 bytes per data frame.
Message Manager	The Message Manager is a part of the Interaction Layer. By the Message Manager the messages received by the CAN bus is offered and the state machine of the Interaction Layer is controlled. It is responsible for the periodic transmission of messages.
Network Database	Database which contains information about a network, including the nodes and the data to be exchanged over the network. The Network Database is used by the Configuration Tool, CANoe, and CANalyzer.
Network Management	By the NM a set of services for monitor the nodes in a network are defined. It is responsible for start-up the network, co-ordination of global operation modes, support of diagnostics, ...
OSEK OS	Specification of an operating system for micro controllers (ECU) especially designed for cars
OSEK COM	Specification of a communication layer for the use with OSEK
Physical Layer	By the Physical Layer all the electrical, mechanical and functional parameters of the connections between network nodes are defined. (ISO

	OSI-Model)
Signal	Variable for data exchange of which the length is defined by the application developer. One or more signals are mapped to a message.
Signal Interface	The Signal Interface is a part of the Interaction Layer. By the Signal Interface the messages offered by the Interaction Layer are split into signals. It is responsible for the combination of signals to messages and the splitting of messages into signals.
Start Delay Time	Time used to delay the beginning of the transmission of a periodic message. The start delay time should prevent transmission bursts caused by interference.
Transmission Mode	Mode to transmit signals or messages. A transmission mode defines whether a message has to be send out periodically or on an event or even in both cases. There are several modes defined to satisfy the needs of the customer’s application.
Transport Protocol	By the Transport Protocol the data longer than a CAN frame is handled. It is responsible for correct splitting and combining data, error detection and error correction. Note: OSEK refers to the “Transport Protocol” as “Network Layer”. They put it in layer 3, not in layer 4 of the ISO OSI Layer Model.

8.2 Abbreviations

Abbreviation	Description
CAN	Controller Area Network
CTS	Component Technical Specification. Platform specific document provided by the car manufacturer. Provides technical details for the component.
ECU	Electronic Control Unit
IL	Interaction Layer
NM	Network Management
OSEK	Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug
TP	Transport Protocol
VN	Virtual Network
VNMF	Virtual Network Management Frame
HLVW	High-Voltage Wake-Up

9 Contact

Visit our website for more information on

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

www.vector.com