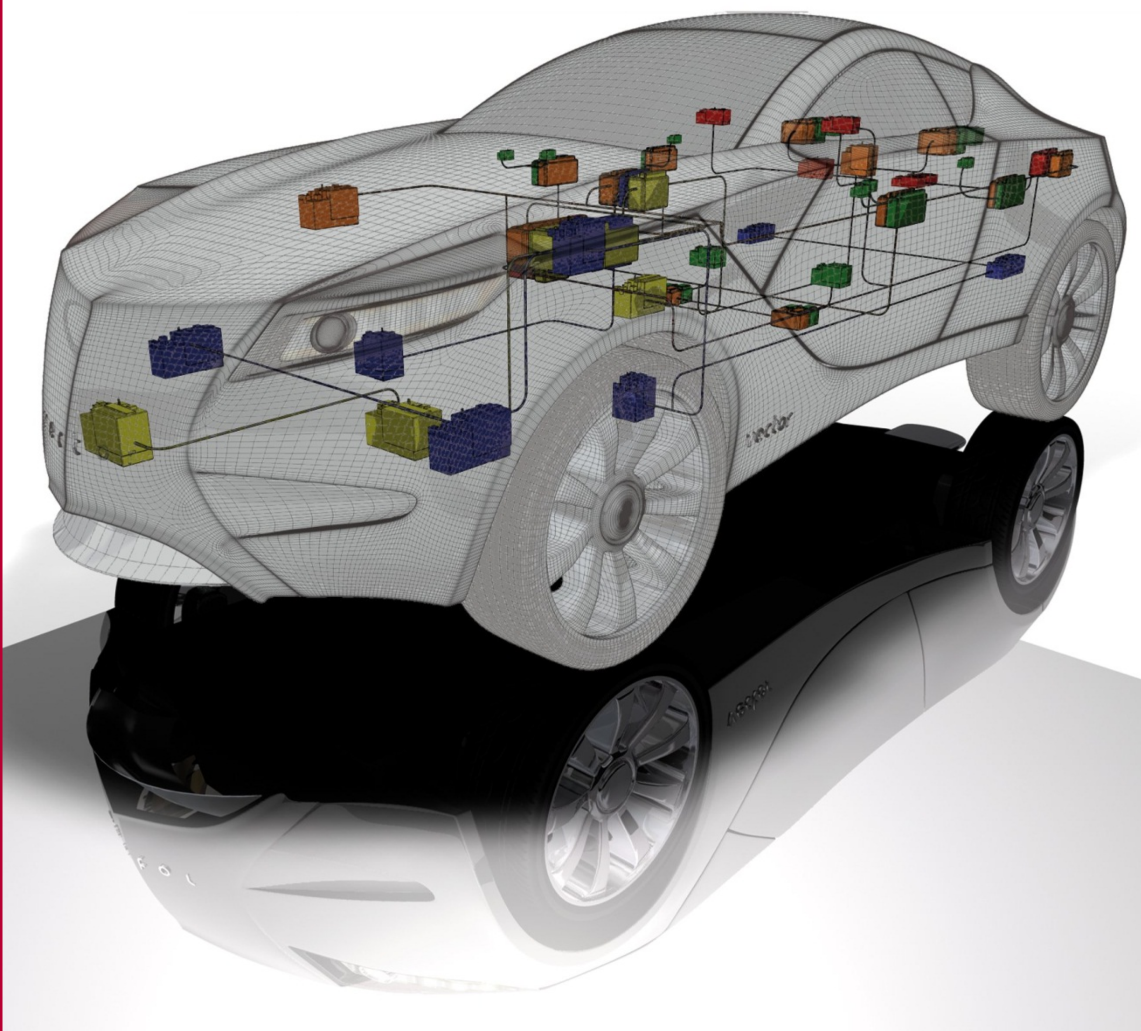


User Manual

GMLAN Package

Installation and usage in CANoe

Version 2.10.0

English

Imprint

Vector Informatik GmbH
Ingersheimer Straße 24

D-70499 Stuttgart

Table of contents

1 Introduction

In this chapter you find the following information:

1.1 Package Overview

This document is the enclosing document for the GMLAN specification 3.0 support package.

Components

This package contains the following components:

- CANoe Interaction Layer GMLAN (GMLAN IL “CANoeILNLGM.DLL”)
- ActiveX message control panel for GM
- GMLAN Network Management simulation (GMLAN NM “GMLAN02.DLL”)
- GMLAN ActiveX NM control panel
- CAPL generator for GM (“CaplGeneratorGMIL.exe”)
- Model Generation Support

Info: This package requires as minimal CANoe version 7.1.

1.2 Reference Documents

[1]	CANoe online help and manual
[2]	Documentation GMLAN NM DLL (CANoe Start Menu – Help)
[3]	Documentation IVLAN NM DLL (CANoe Start Menu – Help)

1.3 Features

The CANoe GMLAN Package supports the GMLAN standard within CANoe. It allows creating automatically a simulation model using the network database, and it supports the Interaction Layer and Network Management functionality within the simulation.

CANoe IL GMLAN

- Provides a signal-oriented interface
- Controls the transmission of messages dependent to the send types of signals and messages.
- Provides the possibility to send messages on request (using a signal or a message as triggering object)
- Provides a fault injection interface to influence the sending of messages

Network Management GMLAN

- The appropriate DLL implements the GMLAN NM protocol that allows nodes that are simulated by CANoe to interact with hardware nodes attached to the CAN network
- Handles all network management concerns for GMLAN including "Virtual Networks" (VNs)

ActiveX message control/GM

- Displays the message- and signal-related information and establishes possibilities to control them

ActiveX NM control/GM	> Displays the network management related information and establishes possibilities to control them in a dedicated way for GMLAN
Main Panel	> Displays all nodes of a configuration and establishes the possibility to open their ActiveX message and NM controls

1.4 History

Package version	Description
2.10.0 of 2013-03-08	Updated CANoe IL CANoeILNLGM.dll to version 2.13.30 > Supporting new message attribute *AppGenMsgCycleTime*: If attribute *AppGenMsgCycleTime* > 0 and *GenMsgSendType* is not ‘cyclicX’ and *GenMsgCycleTime* == 0 and *GenSigCycleTime* == 0, then the message will be set to ‘cyclicX’ with period := *AppGenMsgCycleTime*. Updated GMLAN NM DLL GMLAN02.dll to version 1.9.17 and documentation to version 1.14, cf. [2]: > Supports enabling/disabling of “shared local” VNs by manipulating system variable > Removed function `ILOutput(…)` and added new function `ILOutput2(…)`. Updated "OSEK NM Control" for CANoe 8.0 SP4
2.9.1 of 2012-01-31	Updated CANoe IL CANoeILNLGM.dll to version 2.12.22 > Added function `applILTxPending(long aId, dword aDlc, byte data[])`, cf. section 5.1.10 Updated the GMLAN02 DLL 1.9.14 and documentation to version 1.13, cf. [2] Added the IVLAN DLL in version 2.0.0 with manual [3] and demo configuration
2.9.0 of 2010-10-26	Updated "OSEK NM Control" for CANoe 7.5 Added "Version History" and "Releases and compatibility" section
2.8.0 of 2010-02-12	Release for CANoe 7.2
2.7.0 of 2008-01-30	Release for CANoe 7.0
2.6.0 of 2006-12-01	Release for CANoe 6.0
2.5.0 of 2006-04-05	Release for CANoe 5.2
2.3.0 of 2004-11-12	Release for CANoe 5.1 > Added IL
1.5.5 of 2003-08-01	Release for CANoe 4.1 (NM only)

1.5 Releases and Compatibility

Min. CANoe version

Package name	Package version	Min. CANoe version
GMLAN Package	2.10.0	7.1

Min. CANoe version

Package name	Package version	Min. CANoe version
GMLAN Package	2.9.1	7.1
GMLAN Package	2.9.0	7.1
GMLAN Package	2.8.0	7.1
GMLAN Package	2.7.0	7.0
GMLAN Package	2.6.0	6.0
GMLAN Package	2.5.0	5.2
GMLAN Package	2.3.0	5.1
GMLAN Package	1.5.5	4.1

Caution: Always use the latest possible package version according to your CANoe version!

Min. package version

CANoe version	Package name	Min. package version
>= 8.0 SP4	GMLAN Package	2.10.0
>= 7.5	GMLAN Package	2.9.1
< 7.5 and >= 7.1	GMLAN Package	2.8.0
< 7.1 and >= 7.0	GMLAN Package	2.7.0
< 7.0 and >= 6.0	GMLAN Package	2.6.0
< 6.0 and >= 5.2	GMLAN Package	2.5.0
< 5.2 and >= 5.1	GMLAN Package	2.3.0
< 5.1 and >= 4.1	GMLAN Package	1.5.5

1.6 About this User Manual

1.6.1 Access Helps and Conventions

To find information quickly

The user manual provides you the following access helps:

> At the beginning of each chapter you will find a summary of the contents,
> In the header you can see in which chapter and paragraph you are ((situated)),
> In the footer you can see to which version the user manual replies,
> At the end of the user manual you will find an index, with whose help you will quickly find information.

Conventions

In the two following charts you will find the conventions used in the user manual regarding utilized spellings and symbols.

Style	Utilization
bold	Blocks, surface elements, window- and dialog names of the software. Accentuation of warnings and advices. **[OK]** Push buttons in brackets **File | Save** Notation for menus and menu entries
CANoe	Legally protected proper names and side notes.
`Source code`	File name and source code.
Hyperlink	Hyperlinks and references.
<STRG>+<S>	Notation for shortcuts.

Symbol	Utilization
ℹ	Here you can obtain supplemental information.
⚠	This symbol calls your attention to warnings.
➡	Here you can find additional information.
▲	Here is an example that has been prepared for you.
👣	Step-by-step instructions provide assistance at these points.
✎	Instructions on editing files are found at these points.
🚫	This symbol warns you not to edit the specified file.

1.6.2 Certification

Certified Quality Management System

Vector Informatik GmbH has ISO 9001:2000-12 certification.
The ISO standard is a globally recognized quality standard.

1.6.3 Warranty

Restriction of warranty

We reserve the right to change the contents of the documentation and the software without notice. Vector Informatik GmbH assumes no liability for correct contents or damages which are resulted from the usage of the user manual. We are grateful for references to mistakes or for suggestions for improvement to be able to offer you even more efficient products in the future.

1.6.4 Support

You need support?

You can get through to our hotline at the phone number

+49 (711) 80670-200

or you send a problem report to the CANoe-Support.

1.6.5 Registered Trademarks

Registered trademarks

All trademarks mentioned in this user manual and if necessary third party registered are absolutely subject to the conditions of each valid label right and the rights of particular registered proprietor. All trademarks, trade names or company names are or can be trademarks or registered trademarks of their particular proprietors. All rights, which are not expressly allowed, are reserved. If an explicit label of trademarks, which are used in this user manual, fails, should not mean that a name is free of third party rights.

> Outlook, Windows, Windows XP, Windows 2000, Vista, Windows7 are trademarks of the Microsoft Corporation.

2 Installation and Configuration

In this chapter you find the following information:

2.1 Prerequisites

Operating system	Generation: Microsoft® Windows® 2000, XP, Vista or Windows7 Simulation: Refer the Operating Systems that are supported by CANoe.
CANoe	Version 7.1 (7.1.43) or later must be installed.
User Requirements	The user should be familiar with > Microsoft® Windows® > CANoe If the user intends to do any changes in the simulation model, then the user should be familiar with > CAPL > Interaction Layer functioning > Network Management functioning
Hardware Requirements	Refer CANoe's hardware requirements.

2.2 Installation

Software Installation

Close all applications

Start the setup program - it will guide you through the installation process

Note: Please note that the directory to select is the root directory of your CANoe installation, not the Exec32 directory within this root-directory.

2.3 Update of CANoe

Updating CANoe

The **GMLAN package** extends and/or modifies some basic CANoe components. Thus, the GMLAN package is to be re-installed after CANoe is updated.

Note: Always de-install the **GMLAN package** before you update CANoe, and then install it again.

3 Quick Start

In this chapter you find information about "How to create with the Simulation Model":

3.1 Automatic Generation of the Simulation Model (recommended)

3.1.1 Model Generation by Drag & Drop

1. Prepare a new directory that shall contain the simulation configuration.
2. There, create the folder 'candb' and copy your network database into this folder. The nodes will be generated to the folder "<Database-folder>/../Nodes". The panels will be generated to the folder "<Database-folder>/../Panels".
3. The set-up program has installed a new icon onto your desktop “Create GMLAN configuration”. Open the Windows™-Explorer, drag the network database and drop it onto this icon.

4. A command-line-window is opened and CANoe is started automatically. The Simulation creation runs about ½ up to 2 minutes, depending on your hardware configuration and the complexity of the model. You can track the status of the generation within the write window of CANoe.

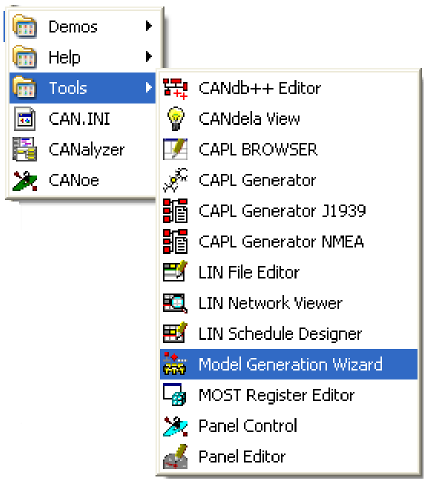
Caution: Please do not operate CANoe before the message

`"GMLAN Simulation creation finished."` has occurred there.

Don't modify any configuration settings, e.g. channel settings, before the model generation is finished.

3.1.2 Model Generation Wizard

1. Prepare a new directory that shall contain the simulation configuration.
2. There, create the folder 'candb' and copy your network database into this folder. The nodes will be generated to the folder "<Database-folder>/../Nodes". The panels will be generated to the folder "<Database-folder>/../Panels".
3. Since the CANoe 5.2 release there is a new model generation tool with a very simple graphical user interface. Start the Model Generation Wizard from the ‘*Tools*’ start menu of the CANoe:



If your CANoe installation is not correctly registered at the system, you will be notified about that and an automated possibility to adjust the registration will be offered.

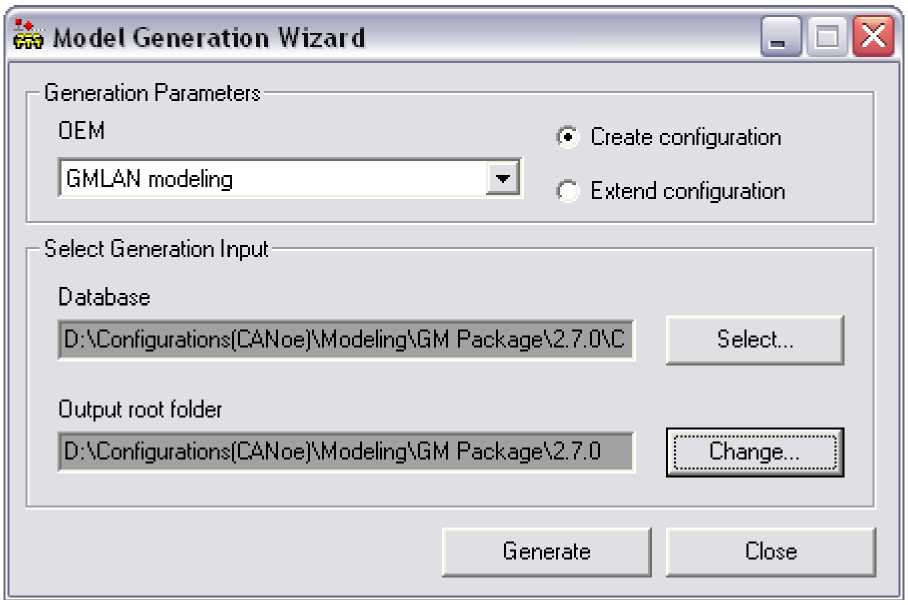
There are a few settings that can be changed before the generation of the model can be started. First, select the OEM "**GMLAN modeling**", then the generation mode:

> **Create configuration:** A new configuration will be created. Please select the network database and verify the output root folder.

> **Extend configuration:** A new bus will be added to the configuration that is currently loaded and active in CANoe. Please select the network database; the output root folder is retrieved from the CANoe configuration automatically.

Note: se the "**Extend configuration"** mode only with configurations that were initially created by the model generation wizard in the "**Create configuration**" mode. Don't modify any configuration settings, e.g. channel settings, before the model generation is finished.

4. Push the button '*Generate*'.



OEM: GMLAN modeling

[✓ Create configuration] Extend configuration

Database: D:\Configurations(CANoe)\Modeling\GM Package\2.7.0\C

Output root folder: D:\Configurations(CANoe)\Modeling\GM Package\2.7.0

CANoe is started automatically. The Simulation creation runs about ½ up to 2 minutes, depending on your hardware configuration and the complexity of the model. You can track the status of the generation within the write window of CANoe.

Caution: Please do not operate CANoe before the message

`"GMLAN Simulation creation finished."` has occurred there.

Don't modify any configuration settings, e.g. channel settings, before the model generation is finished.

Note: " The original database will be not modified. A copy of the database will be created and used for the CANoe simulation. The suffix "Cpy" will be appended to the filename of the network database.

Note: The run of the automatic simulation creation requires the confirmation of the disclaimer of CANoe. Refer the section "COM Server" within CANoe's online-help for details about this topic.

Note: If old CAPL files are found during the generation process, they will be kept.

Note: " If the generation mode "Extend configuration" is used, the new panels are saved in a folder that is named like "…/Panels/<database name>"

Caution: When using the generation mode “Extend Configuration”:

- Use this mode only with configurations that were initially created by the model generation wizard
- Don't modify any configuration settings, e.g. channel settings, before the model generation is finished.

3.1.3 Trouble-shooting

Problems

If the automatic simulation creation fails, then please check the following topics:

Is CANoe currently running? If yes, is the currently active configuration already saved?

If the currently running configuration is not yet saved, then save it.

Do you have multiple installations of CANoe on your system?

During the set-up of the GMLAN Package, do you have selected the CANoe installation area you have installed at last?

In this case please deinstall the GMLAN Package and reinstall it to the CANoe, you have installed at last. If you do not want to use the GMLAN Package within this CANoe installation, then please install the intended CANoe version and then install the GMLAN-package to this CANoe version.

Is your operating system supported?

Refer chapter 2.1 for details.

3.1.4 Result of the Automatic Generation

CANoe configuration	CANoe simulation of all nodes of the given database
CAPL simulation files	Each node of the configuration contains a CAPL file.
Main Panel	Panel with all nodes in order to open their message and NM control panels

3.2 Database Settings – to be Considered before the Simulation Model can be Operated

Network databases for the CANoeIL must fulfil several requirements. Please ensure that you use a GM database with a version that is **at least 7.01** or later.

Binary enumerations such as yes/no are expected always with the negative clause on index 0.

There are several attributes that must be defined to use the database.

3.2.1 Network Attributes

IL

Name	Type	Values	Description
UseGMParameterIDs	Integer	0, 1	Defines if the network uses the GMLAN parameter ID's in the 29-bit architecture. Default: 1

GMLAN NM

Name	Type	Values	Description
VN_<name>	Integer	0 – 255	Defines a unique network ID for the virtual network <name>.
VNInitAct<name>	Enum	0 – No, 1 – Yes	If the value of this attribute is "Yes", the VN <name> will be active on initialization of the system, or on reception of a high voltage wakeup. Otherwise it is in-active and has to be activated explicitly by a node.
NmBaseAddress	Hex	0x620 – 0x63F	Base address for the VNMF messages Default: 0x620
NmMessageCount	Integer	0 – 32	Maximum number of nodes in a network Default: 32
NodeStatusMsgID	Hex	0x0 – 0x1FFFFFFF	ID of the NCA messages Default: 0xFFFE000
NodeStatusMsgCycleTime	Integer	0 – 65535	Periodic rate [ms] of the NCA messages Default: 1200

3.2.2 Node Attributes

CANoe

Name	Type	Values	Description
NodeLayerModules	String	e.g. CANoeILNLGM.dll, GMLAN02.dll	Defines the modules the node needs in the simulation model

IL

Name	Type	Values	Description
SourceId	Hex	0x0 – 0xFF	Source address of a node. Used to build the transmission CAN ID.

GMLAN NM

Name	Type	Values	Description
VNECU<name>	Enum	0 – 4	Role of the node for the specific VN <name>. **Enum** **Mode** 0 None 1 Activator 2 Remoted 3 Activator and Remoted 4 Shared Local
NmStationAddress	Integer	0 – 31	Station address of the node – used by the network management (cf. [2])

3.2.3 Message Attributes

IL

Name	Type	Values	Description
GenMsgILSupport	Enum	0 – No, 1 – Yes	Yes defines the support of the CANoe IL for this message. Default: Yes
NmMessage	Enum	0 – No, 1 – Yes	NM messages are not supported by the CANoe IL. Default: No
TpApplType	String		TP or diagnostics messages are not supported by the CANoe IL.
Prio	Integer	0 – 7	Priority of the message. Used to build the transmission CAN ID. Default: 4
GenMsgSendType	Enum	0, 1	Send type of the message Default: *NoMsgSendType* Due to the signal-oriented approach of the CANoe IL the signal attribute *GenSigSendType* is normally used to define the send types of the simulation model. **Enum** **Mode** 0 cyclicX 1 spontanX Default: *spontanX*

GenMsgCycleTime	Integer	0 – 10000	Cycle time of periodic messages [ms]
AppGenMsgCycleTime	Integer	0 – 10000	If attribute *AppGenMsgCycleTime* > 0 and *GenMsgSendType* is not 'cyclicX' and *GenMsgCycleTime* == 0 and *GenSigCycleTime* == 0, then the message will be set to 'cyclicX' with period := *AppGenMsgCycleTime*.
GenMsgCycleTimeFast	Integer	0 – 10000	Fast cycle time of messages ('IfActive' send types) [ms]
GenMsgStartDelayTime	Integer	0 – 10000	Defines the delay time after start-up of the CANoe IL [ms]
GenMsgDelayTime	Integer	0 – 10000	Defines the minimum delay time between multiple sendings of a message [ms]

3.2.4 Signal Attributes

IL

Name	Type	Values	Description
GenSigStartValue	Integer	0 – 65535	Start value of the signal (raw format) Default: 0
GenSigSendType	Enum	0, 1, 7, 10, 11, 12	Send type of the signal Default: *NoSigSendType* **Enum** **Mode** 0 Periodic 1 OnWrite 7 NoSigSendType 10 OnAnyChange 11 OnChangeIfActive 12 OnDelta
GenSigInactiveValue	Integer		Value of the signal at which it is in an Inactive state. Used by send types '*OnChangeIfActive*'. Default: 0
GenSigCycleTime			Cycle time for a periodic send type
GenSigDeltaValue	Integer		Defines the delta for the send type '*OnDelta*' Default: 0
GenSigSendTopBottom	Enum	0 – 3	Used for the send type '*OnDelta*' to send the value although the delta <

IL

Name	Type	Values	Description
			GenSigDeltaValue **Enum** **Mode** 0 None 1 SendOnTop 2 SendOnBottom 3 SendOnTopBottom Default: *None*

GMLAN NM

Name	Type	Values	Description
VNSig<name>	Enum	0 – No, 1 – Yes	Declares if the signal is assigned to the VN <name>.

Reference: See more about the GMLAN NM attributes in [2].

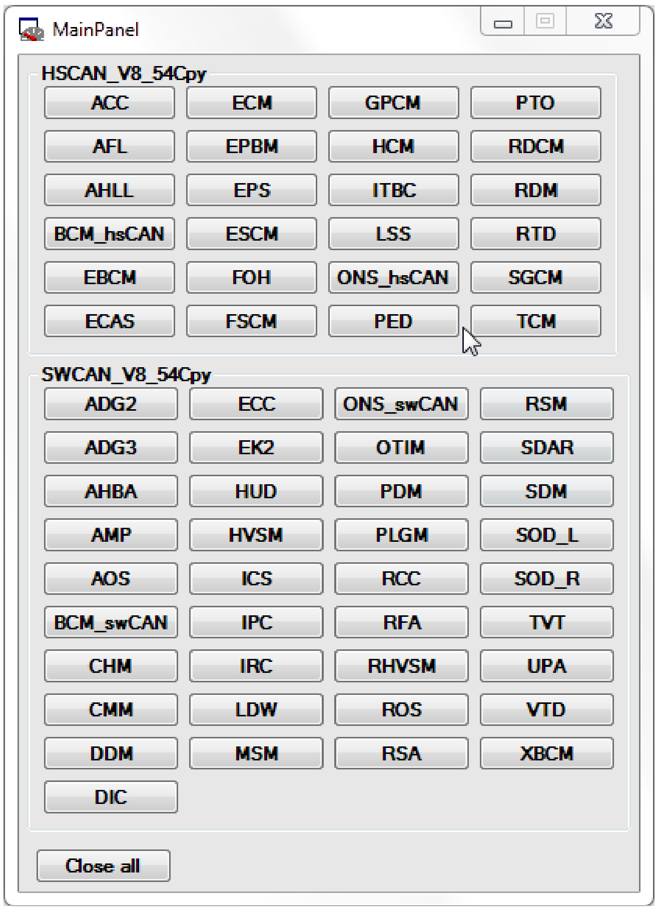
Reference: For possible values of the attribute '*GenSigSendType*' cf. "**CANoe IL | Vector IL | Used Attributes in the Database**" in [1].

4 Operate the Simulation Model

In this chapter you find the following information:

4.1 Main Panel

The generated Main Panel displays all nodes of the configuration. The node buttons opens the ActiveX Message Controls and the ActiveX NM Controls for the appropriate node.



4.2 ActiveX Controls

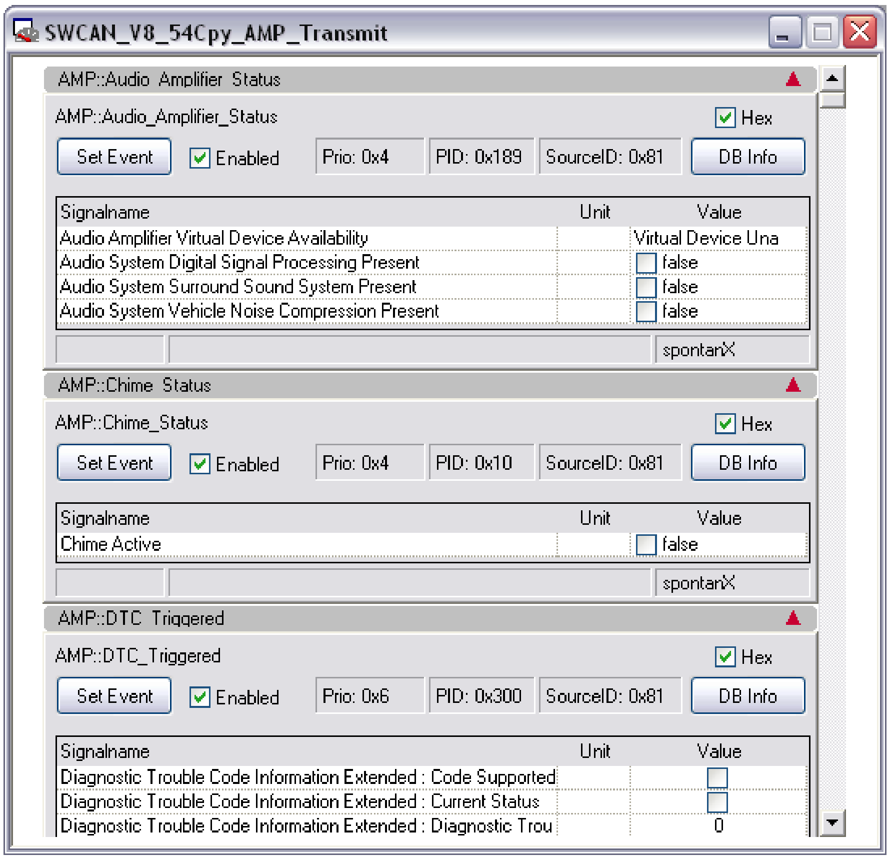
For each simulation node, there are 3 panels available

> “**Send Panel**” - where all transmitted messages are displayed and their mapped signals can be changed

> “**Display Panel**” - where all received messages and received signals are displayed and can be observed. Only the received signals are listed within the received messages.

> **“ECU NM Panel”** - where the status of the Virtual Networks (VNs) can be viewed. This panel additionally allows activating and deactivating the Virtual Networks from the Node's application view for these VNs, the node is an activator.

The send and display panels for the GM package were improved concerning performance and usability issues. They are using the new message group control.



SWCAN_V8_54Cpy_AMP_Transmit

AMP::Audio Amplifier Status

AMP::Audio_Amplifier_Status ✓ Hex

Set Event | ✓ Enabled | Prio: 0x4 | PID: 0x189 | SourceID: 0x81 | DB Info

Signalname	Unit	Value
Audio Amplifier Virtual Device Availability		Virtual Device Una
Audio System Digital Signal Processing Present		☐ false
Audio System Surround Sound System Present		☐ false
Audio System Vehicle Noise Compression Present		☐ false

spontanX

AMP::Chime Status

AMP::Chime_Status ✓ Hex

Set Event | ✓ Enabled | Prio: 0x4 | PID: 0x10 | SourceID: 0x81 | DB Info

Signalname	Unit	Value
Chime Active		☐ false

spontanX

AMP::DTC Triggered

AMP::DTC_Triggered ✓ Hex

Set Event | ✓ Enabled | Prio: 0x6 | PID: 0x300 | SourceID: 0x81 | DB Info

Signalname	Unit	Value
Diagnostic Trouble Code Information Extended : Code Supported		☐
Diagnostic Trouble Code Information Extended : Current Status		☐
Diagnostic Trouble Code Information Extended : Diagnostic Trou		0

Within this group control the user gets control about the visualization of a possible huge amount of messages. The user is able to open and close single message controls on demand.

1. Chose the node you want to observe and press the appropriate button on the main panel.
2. The Network Management Panel shows the current status of the Virtual networks. If a VN is active (for the node) then the rightmost indicator is highlighted. You can activate and deactivate those VNs, the node is an activator.
   At last, you can switch off all sending activities of the node by operating the 'Online'-switch. The node is active per default.
3. Now you can easily set signal values on the send-panels of the node. Each

signal is represented by a value-control you can change. Additionally, each message has a “Set Event”-button available, where an additional transmission can be forced. Normally the IL takes care about the send model itself, but for test purposes, this button can be used. Please note, that the message is sent only, if at least one VN of the mapped signals is currently activated. If not, then a text message is written to the write window of CANoe and no transmission is done.

4. You can additionally observe all messages with their signals, the node receives. The received messages with their signals are listed onto the "display-panels". Please keep in mind, that GMLAN allows the sending of the same message by multiple nodes. Therefore the panels are grouped by the message name and by the sending node. Display panels are not intended to change values; they are intended just to observe the current status.

4.2.1 ActiveX Message Send Control

Preconditions This panel requires a simulated send node where the IL is loaded as node-layer DLL.

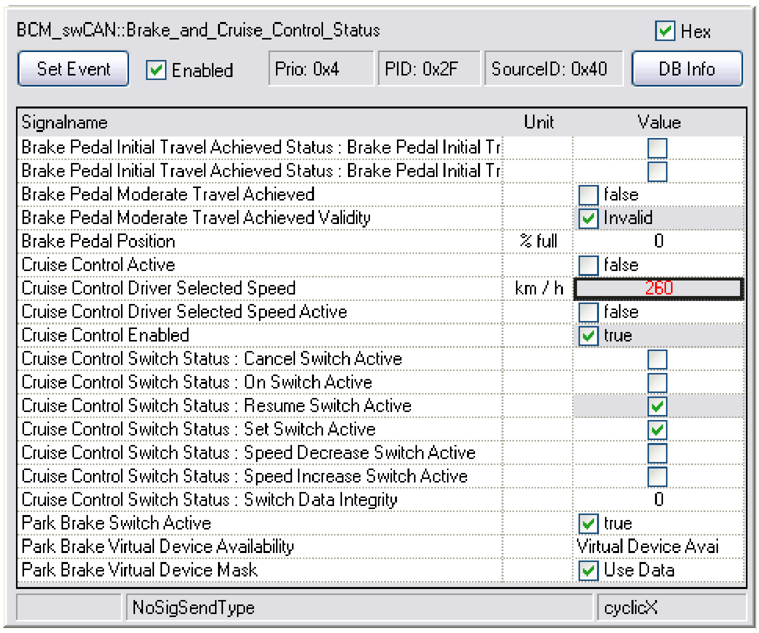
Features The representation of the message ID can be changed between decimal and hexadecimal. The sending can be de-/activated by the checkbox "Enabled".

The enable state of the message can also be changed by the CAPL functions `ILFaultInjectionEnableMessage()` and `ILFaultInjectionDisableMessage()`. Calling those functions also changes this “Enabled” checkbox.

The Button 'DB Info' of the send and display control opens a window that shows the dependency of the virtual networks of the message. This is the resulting dependency after considering all signals of this message.

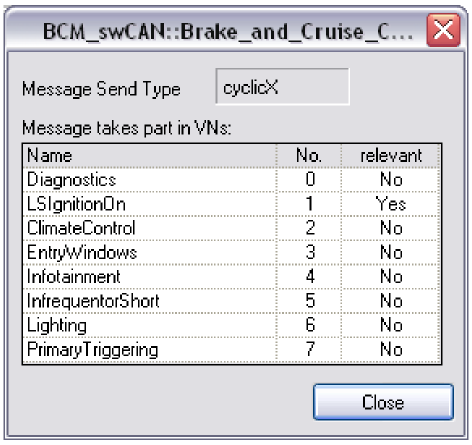
The status line shows the messages send type (right cell) and the signals send type (middle cell) of the currently selected signal.

The send types are taken from the database.



Signalname	Unit	Value
Brake Pedal Initial Travel Achieved Status : Brake Pedal Initial Tr		☐
Brake Pedal Initial Travel Achieved Status : Brake Pedal Initial Tr		☐
Brake Pedal Moderate Travel Achieved		☐ false
Brake Pedal Moderate Travel Achieved Validity		✓ Invalid
Brake Pedal Position	% full	0
Cruise Control Active		☐ false
Cruise Control Driver Selected Speed	km / h	260
Cruise Control Driver Selected Speed Active		☐ false
Cruise Control Enabled		✓ true
Cruise Control Switch Status : Cancel Switch Active		☐
Cruise Control Switch Status : On Switch Active		☐
Cruise Control Switch Status : Resume Switch Active		✓
Cruise Control Switch Status : Set Switch Active		✓
Cruise Control Switch Status : Speed Decrease Switch Active		☐
Cruise Control Switch Status : Speed Increase Switch Active		☐
Cruise Control Switch Status : Switch Data Integrity		0
Park Brake Switch Active		✓ true
Park Brake Virtual Device Availability		Virtual Device Avai
Park Brake Virtual Device Mask		✓ Use Data

	NoSigSendType	cyclicX



Message Send Type: cyclicX

Message takes part in VNs:

Name	No.	relevant
Diagnostics	0	No
LSIgnitionOn	1	Yes
ClimateControl	2	No
EntryWindows	3	No
Infotainment	4	No
InfrequentorShort	5	No
Lighting	6	No
PrimaryTriggering	7	No

The send control provides two possibilities to send a message:

- The Button that shows the message name ignores the transmission model of the database and leads to a message transmission if one of the associated signals takes part at a currently activated VN (analogously to the CAPL function `ILSetMsgEvent(dbMessage msg)` of the Interaction Layer).
- The Input of a value in the column ‘Value’ leads to a transmission of the message dependent on the transmission model of the database (analogously to the CAPL function `SetSignal(dbSignal sig,double value)` or `$Node::Signal = value;`). There are three possibilities to set a signal value:
  - Combo Box with Enumeration values → signals that are defined with a value table in the database
  - Check Box → one bit signals without a value table in the database
  - Decimal Field → signals longer one bit and without a value table in the database

Some features were added to display the signal values:

- The cell of the signal value is greyed means → The current value in the cell is not the value that was received last from the bus.
- The signal value has red colour means → The current value is out of range. The ranges were taken from the database (min/max value of the signal). The ranges are considered only, if at least min or max is different to 0 (zero).

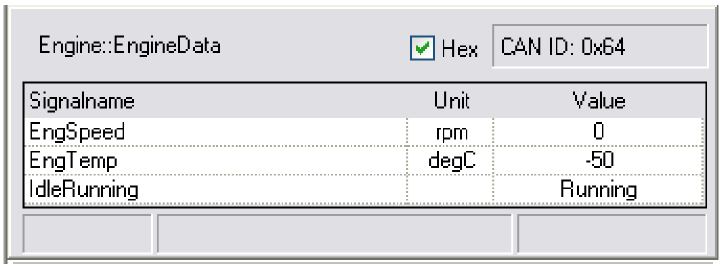
4.2.2 ActiveX Message Display Control

Preconditions None

Features This panel is able to work in both, the simulated and the real environment because it reflects the current signal values.

There are only minor differences between the send and the display control:

- The display control has no 'Set Event' Button, thus no possibility to send
- Signal values are read-only
- The signals of the specific receiver node will be displayed

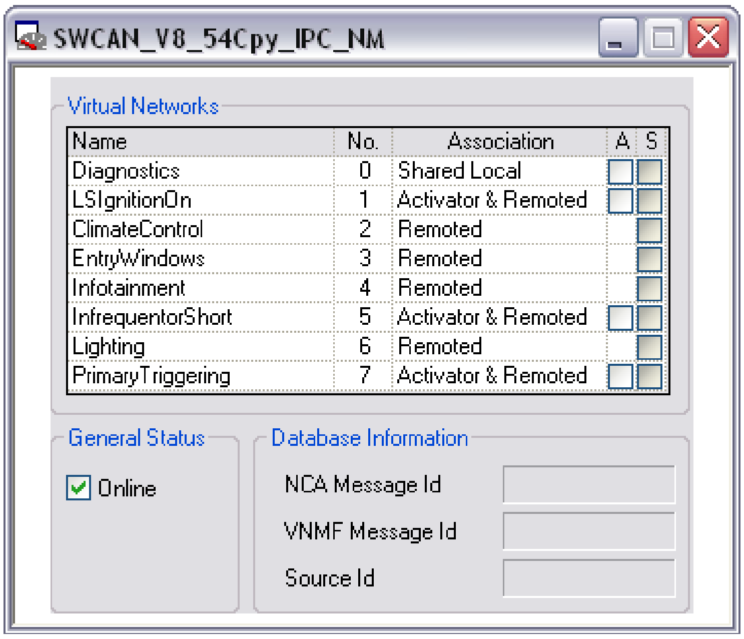

Engine::EngineData ✓ Hex CAN ID: 0x64

Signalname	Unit	Value
EngSpeed	rpm	0
EngTemp	degC	-50
IdleRunning		Running

4.2.3 ActiveX Network Management Control

Preconditions

This control requires the observed node being simulated and the Node-Layer DLLs GMLAN02.dll and CANoeILNLGM.dll loaded into this node.


SWCAN_V8_54Cpy_IPC_NM

Virtual Networks

Name	No.	Association	A	S
Diagnostics	0	Shared Local	☐	☐
LSIgnitionOn	1	Activator & Remoted	☐	☐
ClimateControl	2	Remoted		☐
EntryWindows	3	Remoted		☐
Infotainment	4	Remoted		☐
InfrequentorShort	5	Activator & Remoted	☐	☐
Lighting	6	Remoted		☐
PrimaryTriggering	7	Activator & Remoted	☐	☐

General Status

✓ Online

Database Information

NCA Message Id

VNMF Message Id

Source Id

Features

The network management control shows the status of all relevant virtual networks of the node ('S' for status) and provides the possibility to activate/deactivate all virtual networks ('A' for Application-Activation) with the appropriate association 'Shared Local', 'Activator' or 'Activator & Remoted'.

If a 'Shared Local' network is activated by a network management control, all nodes that are configured as 'Shared Local' for this network will be activated too. As an exception, the diagnostic network (VN number 0) supports only an activation of the single local node.

The information about the NCA, VNMF and Source Id is displayed first after measurement start, because this is runtime information of the GMLAN02.dll.

The Check Box 'Online' establishes the possibility to stop all sending of this node, both from the Interaction Layer and from the Network Management. If one of these two components is inactive, the Check Box will be greyed. This checkbox allows to mute the sending of CANoe IL and NM completely, e.g. for manually executed supervision tests. Please note that the functionality of CANoe IL and NM do not take care about the setting except being silent. After switching a node "offline", the recovery into the online-mode is done roughly just be letting through all further sending. All sending that became due while being 'offline' have been lost.

If it is needed to switch the sending on/off in a more deterministic way (e.g. for reproducible automated tests), then the following API functions are to be used

- `ILControlStop()` and `ILControlStart()` (for CANoe IL) and
- `NwmDiag(…)` (for NM, refer [2] for more details).

4.3 Trouble-shooting

4.3.1 Missing VNMF Messages

Missing VNMF Messages

The current revision of the database (8.54) results in a generated configuration that cannot be operated at once, due to network management reasons: So if the measurement cannot be started with your database, please have a look into the write window of your CANoe. With the mentioned database revision, CANoe complains about missing VNMF Messages. You can recognize problematic node's simulations additionally by the NM address that is shown within the simulation set-up. If there is -1 assigned, then the network management is not able to work for this node.

You can solve this problem by one (or more) of the following means:

- Switch all these problematic nodes into the inactive mode (if the node is not to be simulated)
- Add a valid VNMF-frame into the database and associate it as send message to the node (if the database has to be changed)
- Deactivate network management for these nodes (if the real nodes do not have network management). But please be aware that you then have to activate/deactivate the VNs for the Interaction Layer manually by using the CAPL functions `GMILActivateVN(…)` and `GMILDeactivateVN(…)`.

  To deactivate the network management, you can choose one of the following possibilities:

  1. Do not use the GMLAN02.dll for these nodes. You can deactivate this module within the configuration menu for each (problematic) simulation node, within the tab "modules". Just uncheck the module "GMLAN02.dll"
  2. You can change the database, that this module is not loaded within the node. Just remove the text GMLAN02.dll from the attribute *NodeLayerModules* of each problematic node.

4.3.2 Nodes with the Same Source ID

Multiple defined Source IDs

There is another issue within the current database revision (8.54): There may exist several nodes having the same Source ID. If these nodes are running in parallel within one simulation, then these nodes are working concurrently. This results e.g. in alternating contents of the panels, because the different Interaction Layers are each sending different values. This constellation is to be changed by selecting one node (the right one that is intended for your network setup) for simulation, and deactivating the others.

The CANoe IL detects this case during measurement start. It is indicated by a message within the write window, e.g.

```
Source ID 96 is multiple defined in the database.
```

This message is indicated once for each overlapping. So two occurring messages indicate two additional nodes; that mean that there are 3 nodes of the same Source ID.

5 Interface Specifications

Note: The tool-chain generates a fully working simulation model. You should read this chapter only if there is a need to modify the automatically generated CAPL code.

In this chapter you find the following information:

5.1 Interaction Layer (IL)

Note: These functions (except `ILControlInit()`) should not be used in the "on PreStart()" event procedure.

5.1.1 Interaction Layer Control

The following functions are suitable to control the Interaction Layer. These functions can be used when needed. This may be the case when simulating bus-off-states or network management sleep active states. The standard NM-module (GMLAN02.dll) doesn't need those functions.

Init IL

Syntax	`long ILControlInit ();`
Function	Initializes the IL. If this function is called in "on PreStart()", then the IL will enter the stop state during "on Start()" and must explicitly be started.
Parameter	**None**
Returns	Refer to section 5.1.8

Start IL

Syntax	`long ILControlStart ();`
Function	When the IL is started, then it is fully operating and sending.
Parameter	**None**
Returns	Refer to section 5.1.8

Stop IL

Syntax	`long ILControlStop ();`
Function	Stops sending of messages.
Parameter	**None**
Returns	Refer to section 5.1.8

Suspend IL

Syntax	`long ILControlWait ();`
Function	Stops sending, but accepts signal changes.
Parameter	**None**
Returns	Refer to section 5.1.8

Resume IL

Syntax	`long ILControlResume ();`
Function	Resumes a suspended IL and starts sending messages again.
Parameter	**None**
Returns	Refer to section 5.1.8

Stop IL simulation

Syntax	`long ILControlSimulationOff ();`
Function	Stops the simulation of the IL. The IL is not operational. All API function except function `ILControlSimulationOn()` will not work.
Parameter	**None**
Returns	Refer to section 5.1.8

Start IL simulation

Syntax	`long ILControlSimulationOn ();`
Function	Starts the simulation of the IL. The IL is operational and started. The IL will send messages.
Parameter	**None**
Returns	Refer to section 5.1.8

Please find the state graph that is traversed by these functions in section 5.1.10.

Note: Please note, that send panels require a running IL, otherwise the sending of messages cannot be controlled there.

5.1.2 Setting Signal Values

In order to set a signal value the following assignment should be used:

Set physical signal value

Syntax	`$dbSignal = value;`
Function	The assignment sets the physical value of the signal.
Parameter	**dbSignal:** The symbolic name of a signal in the database. **value:** The pysical value to be set.
Returns	Nothing

Set raw signal value

Syntax	`$dbSignal.raw = value;`
Function	The assignment sets the raw value of the signal.
Parameter	**dbSignal:** The symbolic name of a signal in the database. **value:** The raw value to be set.
Returns	Nothing

Example:

```
$SignalABC = 1;
```

Note: It is recommended to use this assignment to set a signal (with less or equal than 32 Bits) or a floating point signal.

Note: The sending of the mapped messages is dependent to the chosen send types for the message and for the signal that was set.

The following functions can be used for setting signal values directly by the GMLAN IL (obsolete):

Set physical signal value

Syntax	`long ILSetSignal (dbSignal, double value);`
Function	The function sets the physical value of the signal directly at the IL.
Parameter	**dbSignal:** The symbolic name of a signal in the database. **value:** The pysical value to be set.
Returns	Refer to section 5.1.8

Set raw signal value

Syntax	`long ILSetSignalRaw (dbSignal, double value);`
Function	The function sets the raw value of the signal directly at the IL.
Parameter	**dbSignal:** The symbolic name of a signal in the database. **value:** The pysical value to be set.
Returns	Refer to section 5.1.8

Set raw signal byte field

Syntax	`long ILSetSignalRawField (dbSignal, const byte pData[], long aDataLen);`
Function	The function sets the raw value of the signal as a data field. With this function, there is the possibility to set lengthy integers (> 32 Bits).
Parameter	**dbSignal:** The symbolic name of a signal in the database. **pData**: Reference to the input byte array. **aDataLen**: Size of the input byte array.
Returns	Refer to section 5.1.8

Example 1: Setting an integer signal with less or equal 32 Bits or a floating point signal:

```
ILSetSignal(MessageXYZ::SignalABC, 1);
```

Example 2: Setting a signal with more than 32 Bits:

```
void codeQWord2ByteArray (qword value, byte data[])
{
  int i;
  for (i=0; i<8; i++) {
    // Motorola coding:
    data[i] = (value & 0xFF);
    value = value >> 8;
  }
}

on key '2'
{
  qword val = 0;
  byte data[8];

  val += 1;
  codeQWord2ByteArray(val, data);
  ILSetSignalRawField(ECU_APPL_BCM::ECU_APPL_BCM, data,
                      elcount(data));
}
```

Note: If groups of signals shall be set consistently without intermediate inconsistent transmissions, no additional API is needed.

5.1.3 Setting Signal Values - Obsolete API Functions

It is highly recommended always to use the functions that are described within section 5.1.2, that all use the symbolic access by "dbSignal", the functions described in this section are obsolete but still supported.

Set physical signal value

Syntax	`long ILSetSignal (dword sigHandle, double value);`
Function	The function sets the physical value of the signal directly at the IL.
Parameter	**sigHandle:** The signal handle of a signal that must be created prior to the call of this function (see hints below). **value:** The pysical value to be set.
Returns	Refer to section 5.1.8

Set raw signal value

Syntax	`long ILSetSignalRaw (dword sigHandle, double value);`
Function	The function sets the raw value of the signal directly at the IL.
Parameter	**sigHandle:** The signal handle of a signal that must be created prior to the call of this function (see hints below). **value:** The pysical value to be set.
Returns	Refer to section 5.1.8

Set raw signal byte field

Syntax	`long ILSetSignalRawField (dword sigHandle, const byte pData[], long aDataLen);`
Function	The function sets the raw value of the signal as a data field. With this function, there is the possibility to set lengthy integers (> 32 Bits).
Parameter	**sigHandle:** The signal handle of a signal that must be created prior to the call of this function (see hints below). **pData**: Reference to the input byte array. **aDataLen**: Size of the input byte array.
Returns	Refer to section 5.1.8

Signal Handles

Within CANoe 5.0, it is possible to pass signal objects as "Signal Handles". With the earlier version of CANoe (4.1), the Signal Handle had to be produced in CAPL by the programmer. So the programmer had to call the following function to obtain signal handles:

```
dword ILConfigureSignal(char QualifiedSignalName[]);
```

Tip: The function returns over the whole measurement time the same handle for the same signal name. So it may be called multiply or only once after the initialization. To reduce database lookups, it is recommended to call this function only once per signal and node.

The qualified signal name must be enclosed in string quotation marks and may consist of

- the network database name (optional)
- the message name (optional)
- the signal name (mandatory)

If the qualified signal name is not valid respectively to the quotation, it is not possible to obtain a valid signal handle. Then the handle 0 (zero) is returned. If there is interest to gather more hints for the failure, the user may call the following function:

```
long ILErrno();
```

This function returns the error code, if it is called directly after the call of an Interaction Layer API-function.

5.1.4 Reading Signals within CAPL

You can use the functions `GetSignal(…)` to read out the current value of a signal. For GM Extended messages, the signal must be qualified with the node to get the value that was sent last by the node having the dedicated source id. Just have a look into the CAPL browser in the section "Signal Access".

Get physical signal value

Syntax	`var = $dbSignal;`
Function	Retrieves the current physical value of a signal.
Parameter	**dbSignal:** The symbolic name of a signal in the database. **var:** The name of the variable that should store the signal's value.
Returns	Nothing

Get raw signal value

Syntax	`var = $dbSignal.raw;`
Function	Retrieves the current raw value of a signal.
Parameter	**dbSignal:** The symbolic name of a signal in the database. **var:** The name of the variable that should store the signal's value.
Returns	Nothing

The following functions can be used for reading signal values:

Get physical signal value

Syntax	`double GetSignal (dbSignal);`
Function	Retrieves the current physical value of a signal.
Parameter	**dbSignal:** The symbolic name of a signal in the database.
Returns	The physical value of the signal

Get raw signal value

Syntax	`double GetSignalRaw (dbSignal);`
Function	Retrieves the current raw value of a signal.
Parameter	**dbSignal:** The symbolic name of a signal in the database.
Returns	The raw value of the signal

Get last reception time

Syntax	`dword GetSignalTime (dbSignal);`
Function	Retrieves the CANoe time when the signal was on the bus at last.
Parameter	**dbSignal:** The symbolic name of a signal in the database.
Returns	Returns 0 if never

Check range

Syntax	`long CheckSignalInRange (dbSignal, float lowLimit, float highLimit);`
Function	Checks if a signal value is in a dedicated range.
Parameter	**dbSignal:** The symbolic name of a signal in the database. **lowLimit:** lower limit to check **highLimit:** upper limit to check
Returns	Returns 1 if the signal is in the range, returns 0 if the signal is outside the range.

Caution: For GM Extended messages, the signal must be qualified with the node to get the value that was sent last by the node having the dedicated source ID.

Example: Code for Standard-CAN:

```
double value;
value = GetSignal(SignalName);
```

or

```
value = $SignalName;
```

Example code for GM-Extended:

```
double value;
value = GetSignal(NodeName::SignalName);
```

or

```
value = $NodeName::SignalName;
```

5.1.5 Incomplete Transmission Models and Tests

It is possible to ignore the transmit-model in the database and to generate the transmission event manually.

There are two possible levels, where “events” can be injected:

Set signal event

Syntax	`long ILSetEvent (dbSignal);`
Function	This function considers *GenMsgDelayTime*, and it also considers the activity of the network. So a call of this function will lead to a transmission of the associated message, if there is any active VN, the signal is associated to. If there is no active VN, then a message is provided to the write window.
Parameter	**dbSignal:** The symbolic name of a signal in the database.
Returns	Refer to section 5.1.8

Set message event

Syntax	`long ILSetMsgEvent (dbMessage);`
Function	A call of this function will lead to a transmission of the associated message. This function considers *GenMsgDelayTime*. The call will lead to a message transmission if one of the associated signals takes part at a currently active VN.
Parameter	**dbMessage:** The symbolic name of a message in the database.
Returns	Refer to section 5.1.8

Caution: It is strongly recommended, to use this functionality only for test purposes and not in real simulations. The database should be corrected instead.

5.1.6 Fault Injection Functions

Reference: The IL provides a standard fault injection interface which is suitable to control the sending of messages. For more information about these functions refer to the CANoe Online Help [1] "**Technical Reference: CAPL Functions | CANoe IL**".

Generic fault injections

With these fault injection functions the sending behavior of a message can be changed.

Disable message

Syntax	`long ILFaultInjectionDisableMsg (dbMessage);`
Function	Disables the sending of the message except by calling the function `ILSetMsgEvent().`
Parameter	**dbMessage:** The symbolic name of a message in the database.
Returns	Refer to section 5.1.8

Enable message

Syntax	`long ILFaultInjectionEnableMsg (dbMessage);`
Function	Enables the sending of the message.
Parameter	**dbMessage:** The symbolic name of a message in the database.
Returns	Refer to section 5.1.8

Set cycle time

Syntax	`long ILFaultInjectionSetMsgCycleTime (dbMessage, dword value);`
Function	Assigns a new cycle time to the message.
Parameter	**dbMessage:** The symbolic name of a message in the database. **value:** new cycle time in [ms].
Returns	Refer to section 5.1.8

Reset cycle time

Syntax	`long ILFaultInjectionResetMsgCycleTime (dbMessage);`
Function	Resets the cycle time back to its original value from the DBC.
Parameter	**dbMessage:** The symbolic name of a message in the database.
Returns	Refer to section 5.1.8

Set DLC

Syntax	`long ILFaultInjectionSetMsgDlc (dbMessage, dword DLC);`
Function	Sets a new DLC for the message.
Parameter	**dbMessage:** The symbolic name of a message in the database. **DLC:** new data length code.
Returns	Refer to section 5.1.8

Reset DLC

Syntax	`long ILFaultInjectionResetMsgDlc (dbMessage);`
Function	Resets the DLC back to its original value from the DBC.
Parameter	**dbMessage:** The symbolic name of a message in the database.
Returns	Refer to section 5.1.8

Reset all

Syntax	`long ILFaultInjectionResetAllFaultInjections ();`
Function	Resets the fault injection settings for all messages of the node.
Parameter	**None**
Returns	Refer to section 5.1.8

Note: These fault injection functions can also be used for messages with extended IDs (available with CANoe 7.0 SP3). Therefore the ID-based functions have to be used:

```
id_DDM = gmLanId(Driver_Door_Status, DDM);
testDisableMsg(id_DDM);
```

This example disables the sending of the message "Driver_Door_Status" of node "DDM".

Reference: Some fault injection functions are also available in CAPL test modules. Refer to the CANoe Online Help [1] "**Technical Reference: CAPL Functions | Test Feature Set | Fault Injection Functions**".

Caution: It is strongly recommended, to use these fault injections only for test purposes and not in real simulations. The database should be corrected instead.

5.1.7 GM Specific Functions

VN activity control

The following functions are provided to control the activity of virtual networks. If the simulation node uses the Node-Layer module "GMLAN02.dll", then it is not needed to call these functions explicitly, because the Interaction Layer is informed automatically by the network management component about changes in the activity of Virtual Networks.

Activate VN

Syntax	`long GMILActivateVN(long VirtualNetwork);`
Function	Activates the supplied VN.
Parameter	**VirtualNetwork:** The number of the VN
Returns	Refer to section 5.1.8

Deactivate VN

Syntax	`long GMILDeactivateVN(long VirtualNetwork);`
Function	Deactivates the supplied VN.
Parameter	**VirtualNetwork:** The number of the VN
Returns	Refer to section 5.1.8

Check activation state of VN

Syntax	`long GMILIsVNActive (long VirtualNetwork);`
Function	Returns the activity state of the supplied VN.
Parameter	**VirtualNetwork:** The number of the VN
Returns	Returns 1 if yes, 0 if no.

Signal observation

The following function registers a CAPL Handler for a specific signal. It can be used to decide whether a VN should be activated or deactivated. It is possible to provide one handler per signal. On “**setting a signal**” following conditions would lead to a call of the handler:

> Send type "*OnAnyChange*", "*OnChangeIfActive*" and cyclic sending: only if the signal value has changed

> Send type "*OnWrite*": Always

> Send type "*OnDelta*": Only if the delta has exceeded

Register signal observer

Syntax	`long ILRegisterSignalObserver(dbsignal, char handlerName[]);`
Function	Register a call-back function in CAPL as a signal observer.
Parameter	**dbSignal:** The symbolic name of a signal in the database. **handlerName:** The function name of the call-back function. This call-back function must match the following signature: `void nameOfTheSignalObserver(double signalValue)`
Returns	Refer to section 5.1.8

5.1.8 Error Codes

Most of the API-functions are able to return error codes. Some of the API functions when successful return an object that may be de-referenced later on.

In most cases the exact reason of the error is obvious and there is no need to interpret the error code, because the errors are output to the write window additionally. All error messages should contain enough context information to localize the source of the problem.

If there is interest to gather more hints for a failure, the user may call the following function:

Get last error code

Syntax	`long ILErrno ();`
Function	This function returns the last error code, if it is called directly after the call of an Interaction Layer API-function.
Parameter	**None**
Returns	Refer to section 5.1.8

The IL helps CAPL to supply an error message by the transformation of the error code into an error text. The following function can be used for that:

Get error text

Syntax	`long ILSetResultString (long aResult, char aText[], long aLenText);`
Function	This function transforms an error code of the IL into an error text.
Parameter	**aResult:** An error code that is returned by any IL function. **aText:** The output buffer for the text string. **aLenText:** The size of the output buffer.
Returns	Refer to section 5.1.8

Example:

```
long ret;
char buffer[256];
ret = ILSetSignal(MessageXYZ::SignalABC, 1.0);
ILSetResultString(ret, buffer, 255);
```

5.1.9 Definition of Return Codes for the IL

It is not necessary to check the error codes of the API functions. In case of an error the errors are additionally output to the write window.

num. Value (long)	Explanation
0	Request processed
Warnings	Are not reported to the write window
-1	IL is in a state that ignores the given request. → State graph of the IL
-2	Allowed value range of a signal exceeded. (The exceeded value is transmitted to the bus nevertheless)
Configuration errors	Are reported to write window – indicate a fundamental problem that must be solved before usage
-50	Node-Layer is not active. Presumably it is deactivated in the node's configuration dialog.
-51	API function not supported
-52	The node is connected to another bus like the CAN Bus
Dynamic Errors	Are reported to write window. The supplied data is not consumed; therefore it is not needed to set signal's value back.
-100	Signal/message not found in DB, or not mapped to the node
-101	Maximum possible value range exceeded. (There is no transmission to the bus)
-102	A Network management-, diagnosis, or transport protocol signal was set. IL declines such types of signals, because it is not intended to connect these layers to the IL.
-103	Invalid value supplied (for all types of requests)
-104	General error for invalid requests that are not described furthermore.

5.1.10 CAPL Call-back Functions

Background

The CAPL program may define these optional functions (all marked with the prefix `applIL`) to receive indications about events from the IL.

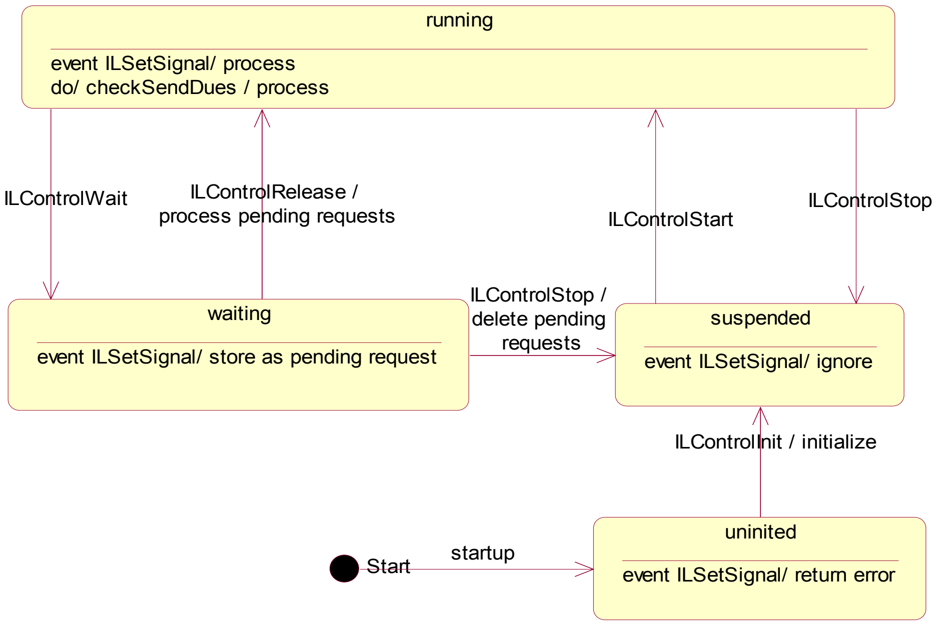
TX pending

Syntax	`dword applILTxPending(long aId, dword aDlc, byte data[]);` `dword applILTxPending(long aId); // deprecated`
Function	This call-back is optionally being called before the IL sends a message to the bus. In this call-back it is possible to prevent the sending of the message or to change the data of the message. An example can be found in section 7.2.
Parameter	**aId:** With the ID you can identify the message. **aDLC:** The DLC of the message to be sent and the length of the data bytes array. **data:** Data bytes array with the bytes to be sent. The bytes can optionally be changed.
Returns	> If the return value of this function is 0, then sending of the message with the supplied `aId` is prevented. > If the return value of this function is 1, then the sending of the message with the supplied `aId` is done.

Caution: You should not use "set signal" functionality within the call-backs, because this may trigger another sending to the bus.

5.1.11 IL States

If the IL is simulated (default), it knows the following states. Please refer the Interface functions for details how to traverse the state graph. These functions are described in section 5.1.2.


running
event ILSetSignal/ process
do/ checkSendDues / process
ILControlWait
ILControlRelease / process pending requests
ILControlStart
ILControlStop
waiting
event ILSetSignal/ store as pending request
ILControlStop / delete pending requests
suspended
event ILSetSignal/ ignore
ILControlInit / initialize
Start
startup
uninited
event ILSetSignal/ return error

Simulation On/Off

If the IL is not simulated, the described functions have no consequences to the IL.

If the IL changes from "SimulationOn" to "SimulationOff", the IL is stopped.

If the IL changes from "SimulationOff" to "SimulationOn", the IL is started.

5.1.12 Write Window Outputs

Here some outputs to the write window are described:

SignalHandle invalid

This signal is not known in the node scope of the IL. Reasons:

> Attribute ‘`GenMsgILSupport`’ of the associated message is not correctly set.
> Send node of the associated message is not the node the IL is responsible for.
> Associated message was identified as a diagnostic, NM or TP message.

MessageHandle invalid

This message is not known in the node scope of the IL. Reasons:

> Attribute ‘`GenMsgILSupport`’ of the message is not correctly set.
> Send node of the message is not the node the IL is responsible for.
> Message was identified as a diagnostic, NM or TP message.

Message for (un)changed signal is not sent

A set signal request doesn’t cause a output of the associated message to the bus. Reasons:

> IL inactive
> Sporadic message send type and no signal send type

Sending of message forced

Only a notification that the message was sent although the send type had prevented a sending. Reasons:

> Usage of “`ILSetEvent`” or ”`ILSetEvent`” in CAPL
> Usage of the send button of an ActiveX send control

5.2 Network Management (NM)

The NM features and interface is described in [2].

NM ↔ IL

There is not implemented any automatic local coordination between the IL and the NM module. The IL and NM in CANoe for GM work independently of each other. Thus, also the CAPL functions for the activation and deactivation of the (local) VN are both present (with different names) at the IL and NM DLL.

ActiveX NM Control

The controls of the Active X NM panel sometime have impact simultaneously onto the NM and IL and sometimes even to a group of nodes (VN) instead of having only local impact to the NM module! (cf. section 4.2.3)

DLL

A NM node’s node attribute *NodeLayerModules* in the database must contain the entry “GMLAN02.dll”.

In opposite to the ActiveX NM Controls all DLL functions (of the IL and the NM) have only impact on the local module (either NM or IL) of that node the CAPL program is attached to.

6 Restrictions and Deficiencies

In this chapter you find the following information:

6.1 Restrictions

Integer signals > 52 bits

The standard message panel-controls that are delivered with this package are working in the following way:

> The setting of values is not possible. (cf. also tip “Sending integer signals > 52 bits” in chapter 7.1)

> The display of values is possible for the maximum value of 2^{52-1}.

7 Tips and Examples

In this chapter you find the following information:

7.1 Tips

Write window views

Make the write window docked so that other windows will not hide it. Sometimes it is crucial to cf. the write window because all error messages and warnings will be output there.

Setting a signal at the IL

If you use the function `ILSetSignal()` make sure that you use it in the correct node. I.e. if you want to set a ESP signal value you must edit the “ESP.can” file, otherwise you will see no change to the signal value. This is a correct behaviour of the IL because one node cannot set a signal value of another node. Therefore it is recommended to use always the `SetSignal()` function. Then the routing to the correct node is done automatically.

Sending integer signals > 52 bits

If the standard mechanism (refer 5.1) does not match the requirements, then an additional API function of GM CANoe IL can be used. With this function, the **raw value** can be set. It is not possible to set the physical value of such signals within CAPL.

1. Create an environment variable of type 'data' with the intended length within the network database (or an additional database you have to associate to the used channel(s)).
2. Put a Hex-Edit-control onto a (new) panel and associated it to environment variable that was created in the previous step.
3. Open the CAPL program of the send node of the signal, add an "on envVar"-handler for that environment variable and add the following code fragment:

```
on EnvVar <myEnvvar>
{
  byte lValue [8];
  GetValue (this, lValue);

  ILSetSignalRawField(<Message>::<Signal>, lValue, 8);
}
```

Then replace the names in brackets:

> `<myEnvvar>` by the previously created environment variable
> `<Message>` by the message that contains the lengthy signal
> `<signal>` by the lengthy signal.

Please note, that all these names are to be inserted without any quotation.

There are some rules to follow when operating these hex-edit controls:

> The first byte to enter is the lease significant byte of the raw value of the signal.
> Only those bytes are affected that are supplied in the control. If for example the last byte is omitted, then this last byte remains unaffected! ' It is recommended to supply always all needed bytes.

It is currently not possible to display these lengthy integers on the panels:

Caution: The displayed values represent only a part (52 bits) of the really received value. Recommendation not to consume the value from the display panel. Use the trace window instead.

Get hooks on activation/deactivation of Virtual Networks

The network management component communicates automatically with the Interaction-Layer component and automatically activates and deactivates the VNs. If the activation/deactivation of the Virtual Networks should be observed, then it is possible to define CAPL functions that are called on activation/deactivation of the Virtual Networks. See [2] and 5.1.7 for details.

Avoid the automatic start of the Interaction Layer

The Interaction Layer is started per default on start of measurement. If this should be prevented, then call the initialization function (cf. 5.1.1) of the CANoe IL in the pre-start section manually. Then the automatic start is not done. Please note, that send panels require a running CANoe IL, otherwise the sending of messages cannot be controlled there.

Get application callbacks (hooks) on transmit activity of the Interaction Layer

It is possible to get an application call-back within CAPL before the GMLAN IL sends a message to the bus. In this call-back it is possible to prevent the sending of the message or to change the data of the message.

```
dword applILTxPending(long aId, dword aDlc, byte data[]);
dword applILTxPending(long aId); // old version
```

> Cf. section 5.1.10 for details.

Caution: You should not use "set signal" functionality within the call-back, because this may trigger another sending to the bus.

CAPL signal driver

If a signal value will be set (e.g. in a panel, by the COM-API, within CAPL) a signal driver is used to set the value and send the message. Normally the GMLAN IL is the signal driver of all TX signals within an ECU. But for dedicated signals or for test purposes it might be helpful to send the message within CAPL. Therefore a CAPL signal driver (CAPL call-back) can be registered, that is called on each set signal request.

```
long registerSignalDriver(dbSignal, char callback[])
```

The CAPL call-back must have following signature: `void fct(double value)`

How to globally activate a VN from CAPL that is locally “shared local”?

All CAPL function for the activation or deactivation of the VN has only impact on the local node. Especially al nodes that are “shared local” to this VN can only be activated in their local CAPL program. In order to trigger all nodes from one dedicated node a new trigger event must be defined, that is handled in all participating nodes accordingly.

1. Create an environment variable of type 'integer' with name VN_Act_<name> for the VN <name> with the ID *n* within the network database (or an additional database you have to associate to the used channel(s)) of the CANoe configuration..
2. Open the CAPL programs of all participating nodes, add an "on envVar"-handler for that environment variable and add the following code fragment:

   ```
   on EnvVar VN_Act_<name>
   {
     long i;
     i = <n>;
     if (@this == 0)
     {
       GMILDeactivateVN(i); // Deactivate VN at IL
       ILDeactivateVN(i); // Deactivate VN at NM
     }
     else
     {
       GMILActivateVN(i); // Activate VN at at IL
       ILActivateVN(i); // Activate VN at NM
     }
   }
   ```

   Then replace the names in brackets:

   > `<name>` by the name of the VN

   > `<n>` by the unique ID of the VN (equal to value of database network attribute *VN_<name>*)

   Please note, that all these names are to be inserted without any quotation.
3. In the local node’s CAPL program that should activate or deactivate the appropriate VN simply set the value of the environment variable:

   ```
   @VN_Act_<name> = 1; // 1 to activate or 0 to deactivate
   the VN
   ```

Instead of using an environment variable also a system variable of CANoe can be used.

7.2 Examples

Example 1: Individual calculation of a checksum and a message counter

```
dword applILTxPending (long aId, dword aDlc, byte data[])
{
  dword i;
  byte xor;

  /* Message 0x1A0 contains a XOR checksum in Byte 0. It will
   * be calculated from the other data bytes of the message.
   */
  if(aId == 0x1A0)
  {
    // calculate
    xor = 0x00;
    for(i = 1; i < aDlc; ++i) {
      xor = xor ^ data[i];
    }
    // set the new checksum
    data[0] = xor;
  }

  /* Message 0x1A1 contains a 4-Bit message counter in
   * the first 4 Bits of Byte 0.
   */
  if(aId == 0x1A1)
  {
    // get the old value
    i = data[0] & 0x0F;

    // increment
    i++;
    i = i % 16;

    //set the new message counter
    data[0] = i & 0x0F;
  }

  return 1; // don't prevent sending of the message
}
```

Note: Use the new fault injection interface (4.1.6) to prevent the sending of a message.

8 Index

Get more Information!

Visit our Website for:

> News

> Products

> Demo Software

> Support

> Training Classes

> Addresses

www.vector.com