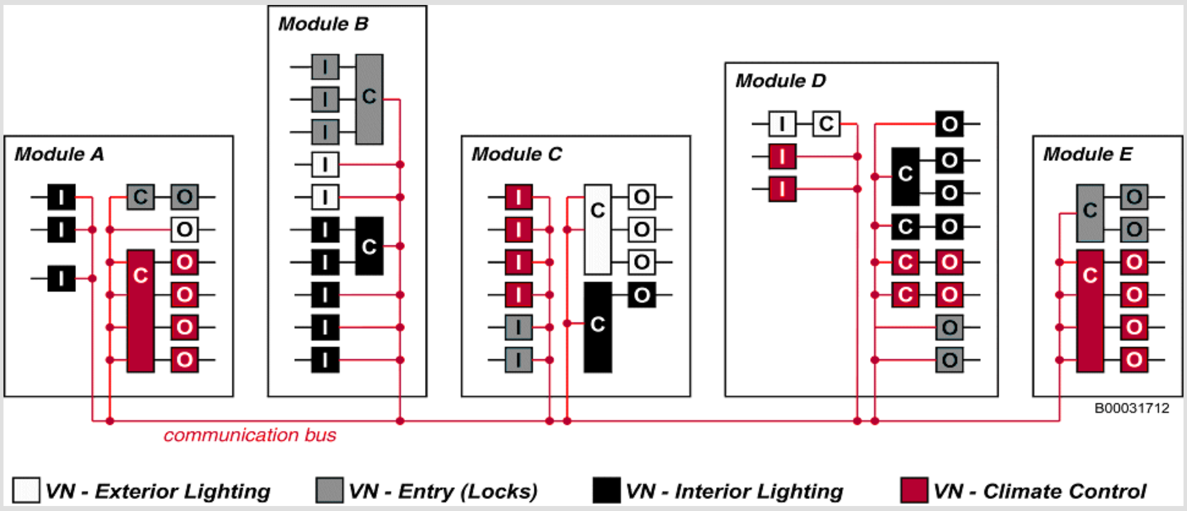
Startup with GMLAN and GENy

User Manual

(Your First Steps)

Version 1.0.1

Vector Informatik GmbH, Ingerheimer Str. 24, 70499 Stuttgart
Tel. 0711/80670-0, Fax 0711/80670-399, Email can@vector-informatik.de
Internet http:\\www.vector-informatik.de



Authors:	Klaus Emmert
Version:	1.0.1
Status:	in preparation (in preparation/completed/inspected/released)

History

Author	Date	Version	Remarks
Klaus Emmert	2008-03-10	1.0	Created
Klaus Emmert	2011-11-11	1.0.1	Improved Understandability

Motivation For This Work

You want to read a document and follow its installation hints and as a result the software is basically running? Then continue reading this user manual.

It is a guide through the installation of the GMLAN components, helps you to configure you application for the needs of GMLAN and gives you precious hints for the different requirements an ECU using GMLAN has to fulfill (PowerTrain, SingleWire CAN…).

WARNING

All application code in any of the Vector User Manuals are for training purposes only. They are slightly tested and designed to understand the basic idea of using a certain component or a set of components.

Contents

Illustrations

1 About this Document

This document gives you an understanding of GMLAN. You will receive general information, a step-by-step tutorial to get the Startup with GMLAN running.

1.1 Legend and Explanation of Symbols

You find these symbols at the right side of the document. Use this helpful feature to find fast the topics, you need information about.

These areas to the right of the text contain brief items of information that will facilitate your search for specific topics.

Symbol	Meaning
▲■	The building bricks mark examples.
Comments and explanation	You will find key words and information in short sentences in the margin. This will greatly simplify your search for topics.
👣	The footprints will lead you through the steps until you can use the Startup with GMLAN.
⚠	There is something you should take care about.
ℹ	Useful and additional information is displayed in areas with this symbol.
✎	This file you are allowed to edit on demand.
🚫	This file you must not edit at all.
?	This indicates an area dealing with frequently asked questions (FAQ).

2 What is GMLAN

GMLAN is CANbedded for GM

GMLAN is General Motors' network operating software, which is supplied exclusively by Vector. Vector has great experience with GMLAN from its beginning on since 1999. Used by GM and its subsidiaries worldwide, GMLAN provides a multi-layered network software solution for CAN bus management and access.

GMLAN consists of mandatory components and optional components.

Vector provides all current GMLAN software implementations and each software implementation is:

- Microcontroller-specific
- CAN controller-specific
- Compiler-specific

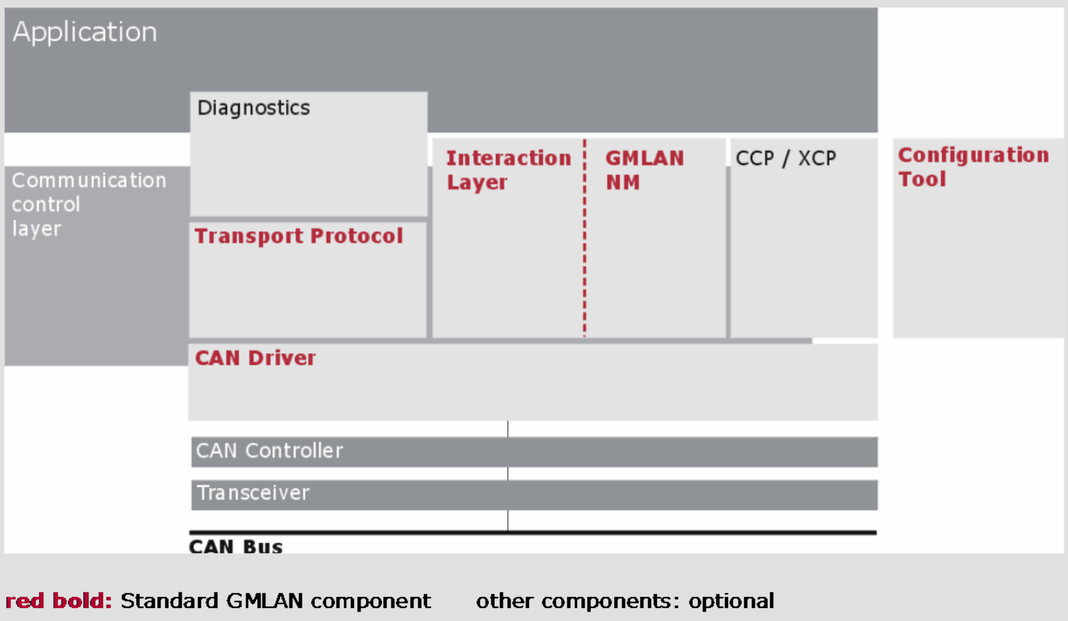


Figure 2-1 GMLAN Layers with standard and optional components

Mandatory components of GMLAN

CAN Driver

The CAN Driver interconnects to the CAN controller, manages CAN transmission and reception, handles CAN initialization, bit rate, sample point, SJW value and filtering. The CAN Driver interfaces with the GM-specific network software and provides a common low-level interface across a variety of CAN controllers.

Interaction Layer

It handles message reception, transmission, event and periodic messages, manages Virtual Network initialization and fault detection (signal faults, VN faults).

Transport Protocol

The Transport Protocol implements the requirements of GMLAN network layer and is primarily used for diagnostics. It manages large data transfers using USDT with data size up to 4095 bytes. The Transport Protocol is based on ISO 15765.

GMLAN Network Management

It handles both node and network management, manages communication states (active, enabled, sleep). It manages node states (power up, sleep, wake up) and the virtual networks. It also handles the use of the VNMF message.

Configuration Tool

It is PC-based and operates with Windows 2000/XP. The configuration tool is delivered with the GMLAN Software Components and is used to scale the software down to your specific needs, to optimize the use of ROM / RAM resources and to select the GMLAN features that require integration into your ECU.

Info
The CAN Driver and the Transport Protocol are standard components and are not GM-specific. The main difference of the GMLAN software stack in comparison with other OEM-specific stacks is the Interaction Layer / GMLAN Network Management and the concepts of the virtual networks.

Additional components, optional available:

Diagnostics - CANdesc HIGHLY RECOMMENDED by GM!

This is a functional interface for diagnostic services based upon the CDD diagnostic data base created by CANdelaStudio. It fulfills the requirements of GMW3110, V1.5 and handles direct processing of CAN-specific diagnostic requests (enable/disable normal message transmission), negative responses (e.g. service not available), exceptions (e.g. busy, request pending) and address handling (detection of response service identification). It also supports UUDT-message transmission. Testing of diagnostic is easy using CANdito, CANoe or CANape

Communication Control Layer CCL

It supports integration of the software components CAN Driver, Interaction Layer, Network Management, Transport Protocol and Diagnostics.

Also it provides an abstraction for different vehicle manufacturers, microcontrollers, compiler/linker, CAN controllers / transceivers. It supports a global debug mechanism and is configured via the configuration tool.

Measurement and Calibration (CCP/XCP)

The CCP/XCP is a high performance CAN-based monitor program. It supports memory read / write and calibration activities and provides data acquisition. It can also be used for flash programming and data security. It works with Vector’s CANape tool.

2.1 Virtual Networks

In general the ECUs form a network where every ECU is a network node.

The concept of GMLAN Network Management groups distributed functions into so-called Virtual Networks (VNs).

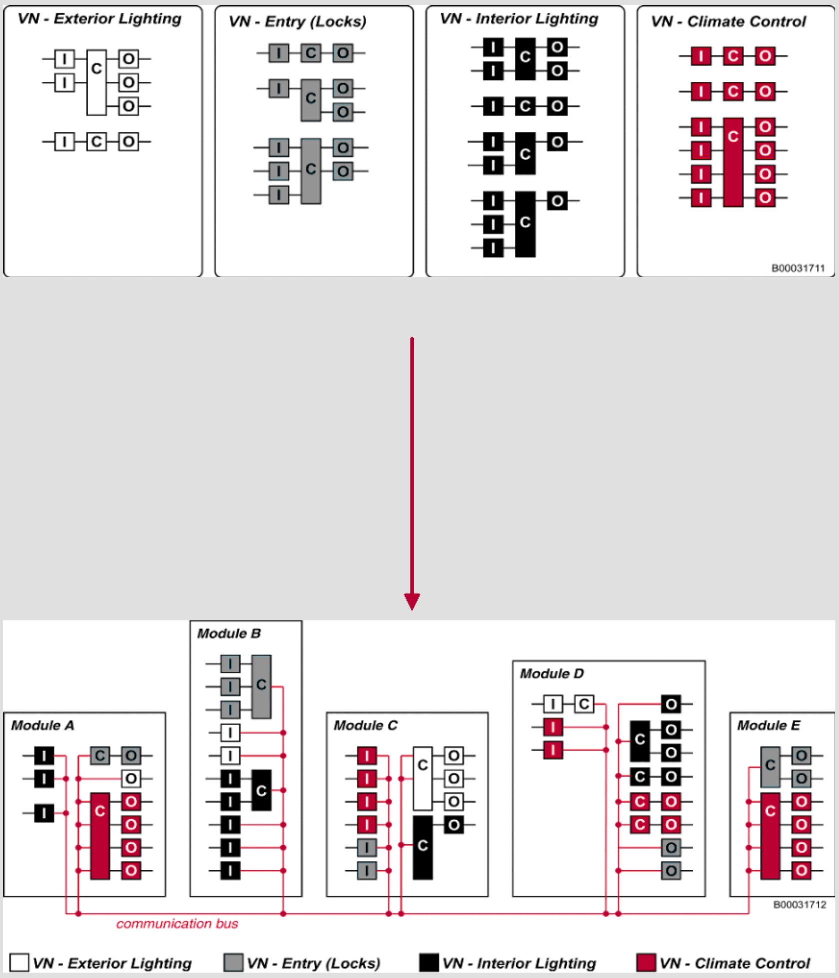


Figure 2-2 Virtual Networks

What comes along with this concept?

- Each VN is a functional entity of GMLAN.
- The location of a VN or parts of it is independent of the physical partitioning

- The VNs are spread over the different ECUs. An ECU can only go to sleep mode, if all VNs inside are asleep.
- Each VN handles its related signals, the use of the network and may include sleep/wakeup.
- Related signals are grouped according to their function
- A message can contain signals of different VNs. There must be a decision whether a signal of a VN is valid (VN awake) or not (VN sleeps).

3 Your ECU

Before you start with implementing GMLAN Software Components take a look at the ECU you have to develop regarding the following questions:

❶ Is it a **Body Bus** ECU? With Single Wire CAN and mixed IDs (11bit and 29bit) or 11 bit IDs only? Is it a multiple Identity ECU?

❷ Is it an **Infotainment** ECU?

❸ Is it a **Powertrain** ECU?

The above mentioned kinds of ECUs and bus systems have many settings in common and differ in special settings.

The following installation description shows typical configurations including the differences for the three mentioned cases.

3.1 Body Bus ECU

Single Wire CAN Mixed Identifiers (11bit and 29bit)

- Typically 33.333 kBaud
- 1 wire (bus communication possible while other ECUs are in sleep mode)
- 2 initialization objects predefined: 33.333 kBaud and 83.333 kBaud for flashing
- 29 bit application messages
- Source Learning
- Flexible monitoring of receive messages

Single Wire CAN 11 Bit Identifier

- Typically 33.333 kBaud
- 1 wire (bus communication possible while other ECUs are in sleep mode)
- 2 initialization objects predefined: 33.333 kBaud and 83.333 kBaud for flashing

Multiple ECU on Single Wire CAN

- Used for nodes that exist more than once in a car with only a few differences
- At power up the application decides by means of a function call (see below), which node should be realized, e.g. left passenger door, or right driver door.
- To reduce the memory consumption messages that are sent exclusively from one node can be overlapped with the exclusively sent messages from the other nodes. The result of this overlapping is that all these messages share a

common memory location for the transmit data. This can be configured manually in the **Send message** tab in the Configuration Tool.

3.2 Infotainment ECU

- CAN Class C
- 125 kBaud
- 2 wires
- Able to be woken up
- Mixed IDs

3.3 PowerTrain ECU (or Chassis expansion or Powertrain expansion)

- CAN Class C
- 500 kBaud
- 2 wires
- Clamp 15 with and without follow-up time
- Shared local-input activated VN
- Special bus off recovery time

4 GMLAN In 6 Steps

STEP 1 : PREPARE YOUR SOFTWARE PROJECT

Create folders in your application to store the delivered source code components and the generated files in.

STEP 2: CONFIGURATION TOOL AND DATA BASE FILE

Read-in the data base files that should be provided from your OEM (DBC file / CDD file for diagnostics) in the Configuration Tool and make the configuration settings for the CANbedded Software Components.

STEP 3: ADD FILES TO YOUR APPLICATION

Add the CANbedded C and H files to your project or makefile.

STEP 4: ADAPTATIONS FOR YOUR APPLICATION

Now your application files must be modified to use the CANbedded Software Components (includes, cyclic calls, initialization) and do the call back functions.

STEP 5: COMPILE AND LINK

Compile and link the complete project and download it to your test hardware or development environment.

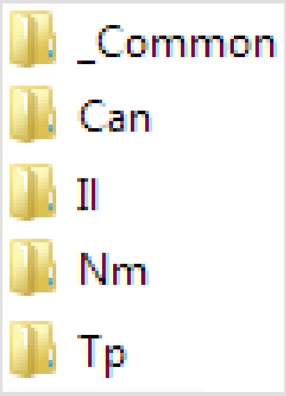
STEP 6: TEST THE SOFTWARE COMPONENT

Test the software via a suitable test environment.

4.1 STEP 1 Prepare Your Software Project

Following the GM_Readme document you have already installed the delivered GMLAN Software Implementation package. To use the GMLAN components for your application copy the needed component source code into your project folder.

Normally these are the following component folders:


_Common
Can
Il
Nm
Tp

This includes the code for the CAN Driver, the Interaction Layer, the Network Management (GMLAN) and the Transport Protocol.

Info
For the diagnostic there is an additional software component of Vector Informatik available - CANdesc. This component is completely generated; you won't find any source code in your delivery.

GenData

A folder **gendata** is optional but recommended to be the target for all the data that is generated by the Configuration Tool.

4.2 STEP 2 Configuration Tool GENy and DBC File

Info
Use the online help of the configuration tool GENy for more information about using GENy and its concepts. You additionally find a comparison between GENy and CANgen.

Start the Configuration Tool GENy.exe. Select pre-configuration file, microcontroller, derivative and compiler in the first **Setup Dialog** window.

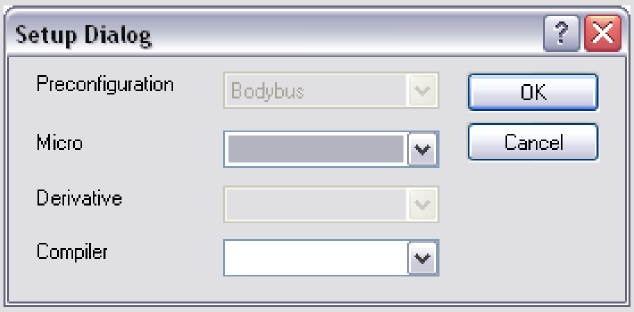


Figure 4-1 Setup Dialog

The **Preconfiguration** selection determines which settings are preset in GENy and which settings are not shown at all. This makes configuration for you very easy.

There is available:

- Bodybus
- Infotainment
- Powertrain

Then go on with adding a data base file using the green plus

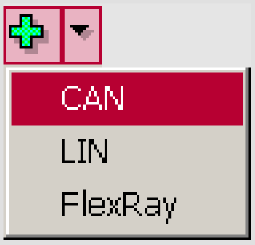
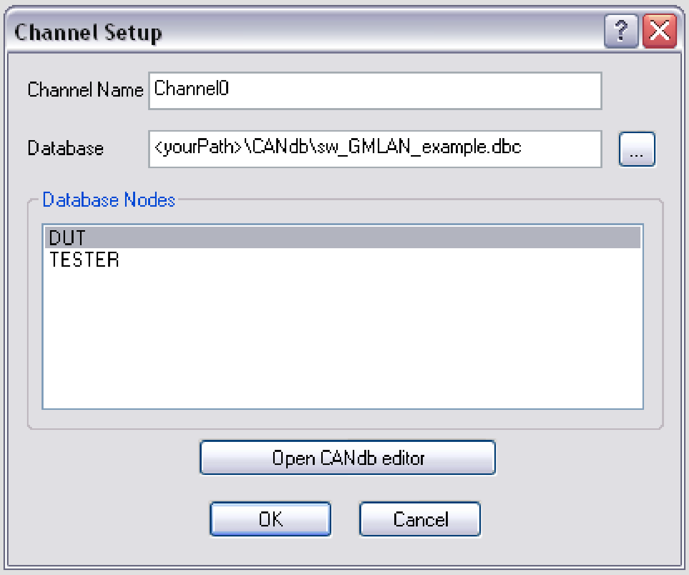


Figure 4-2 Add data base file for a certain bus system, here CAN.

Info
The DBC file is designed by the vehicle manufacturer and distributed to all suppliers that develop an ECU. Thus every supplier uses the SAME DBC file for one vehicle platform and one bus system (powertrain, body CAN etc.) to guarantee a common basis for development.

The DBC file contains e.g. information about every node in the network, the messages/signals each node has to send or to receive. The distribution of the signals among the messages is stored in the DBC file, too.

Use the browse button **[…]** to select your data base file (DBC).



Channel Name: Channel0

Database: <yourPath>\CANdb\sw_GMLAN_example.dbc

DUT
TESTER

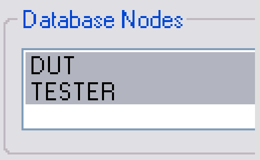
Figure 4-3 Channel Setup Window

As you will use a different data base file you will get more and other data base nodes displayed.

The Nodes list shows all available ECUs within the CAN network. Select yours.

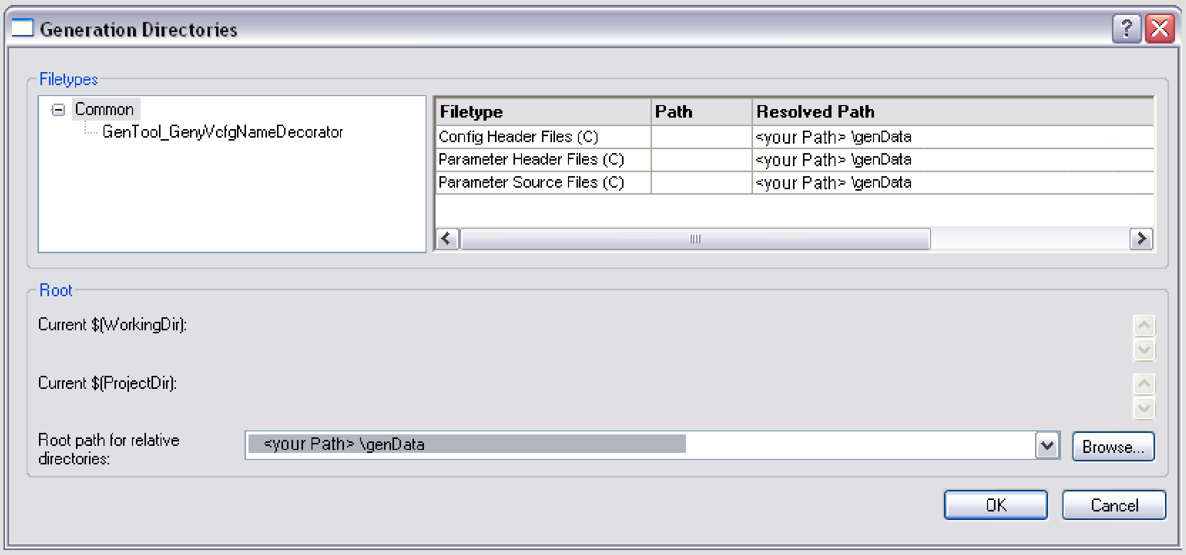
Caution for Multiple ECUs
Use the <Strg> key to mark more than one ECU in the Nodes list box then the Multiple Nodes section will be accessible.

Confirm with **[OK]**.



4.2.1 Setting of Generation Paths

Open the menu **Configuration|Generation Paths…** to set the paths where the Configuration Tool has to generate the files in. Here we use the folder **gendata** to collect all generated files.



Filetype	Path	Resolved Path
Config Header Files (C)		<your Path> \genData
Parameter Header Files (C)		<your Path> \genData
Parameter Source Files (C)		<your Path> \genData

Current $(WorkingDir):

Current $(ProjectDir):

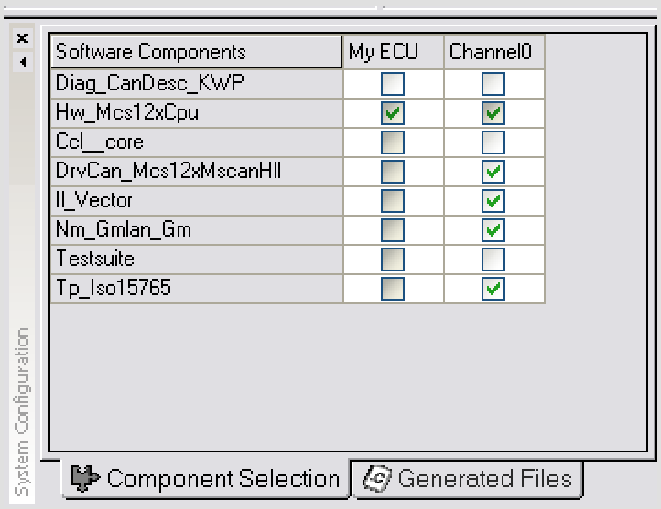
Root path for relative directories: <your Path> \genData

Figure 4-4 Paths To Generate Files to destined folders

Enter your **Root Path for relative directories**. Additionally you can add extra path extensions below the **Path** column. Always make sure that the **Resolved Path** points to the correct path!

4.2.2 Component Selection

Use the Component Selection to choose all components you need for your project. The screenshot below shows a possible selection.



Software Components	My ECU	Channel0
Diag_CanDesc_KWP	☐	☐
Hw_Mcs12xCpu	✓	✓
Ccl__core	☐	☐
DrvCan_Mcs12xMscanHll	☐	✓
Il_Vector	☐	✓
Nm_Gmlan_Gm	☐	✓
Testsuite	☐	☐
Tp_Iso15765	☐	✓

Figure 4-5 Component Selection

4.2.3 Tree view – A List of all selected components

All components selected before are now displayed in the **Tree View** below **MyECU|Components**.

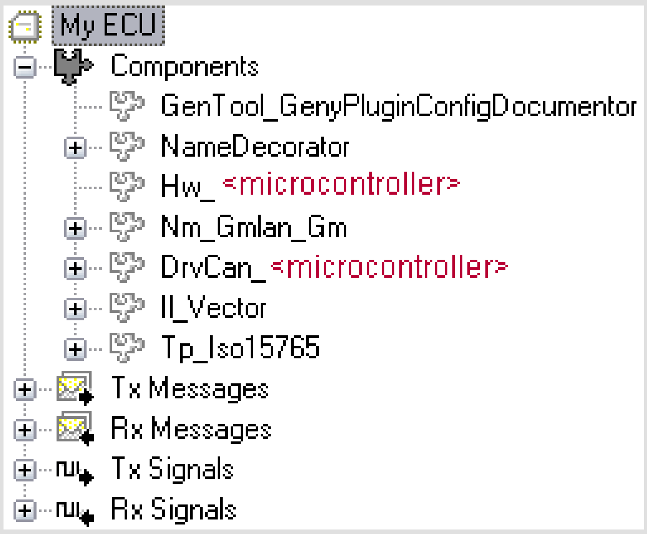

My ECU
Components
GenTool_GenyPluginConfigDocumentor
NameDecorator
Hw_ <microcontroller>
Nm_Gmlan_Gm
DrvCan_<microcontroller>
Il_Vector
Tp_Iso15765
Tx Messages
Rx Messages
Tx Signals
Rx Signals

Figure 4-6 Tree View of all selected components

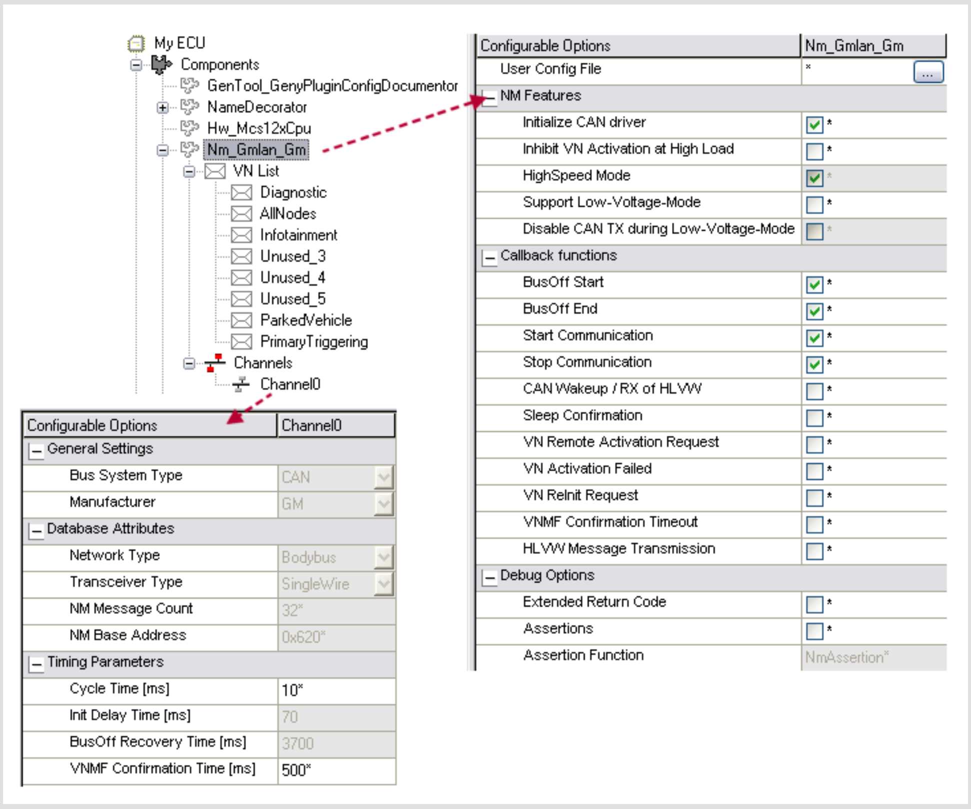
Now select the different components to perform their individual configuration in their **Configuration View** right besides the **Tree View**.

4.2.4 HW_<microcontroller>

This contains the hardware specific CAN driver settings for your controller. Refer to the manual TechnicalReference_CANDriver_<hardware>.pdf and to your controller manual for more information.

4.2.5 Nm_Gmlan_Gm

These settings are derived from the data base file. For this first start-up you can work with the default settings.



- My ECU
  - Components
    - GenTool_GenyPluginConfigDocumentor
    - NameDecorator
    - Hw_Mcs12xCpu
    - Nm_Gmlan_Gm
      - VN List
        - Diagnostic
        - AllNodes
        - Infotainment
        - Unused_3
        - Unused_4
        - Unused_5
        - ParkedVehicle
        - PrimaryTriggering
      - Channels
        - Channel0

Configurable Options	Channel0
General Settings	
Bus System Type	CAN
Manufacturer	GM
Database Attributes	
Network Type	Bodybus
Transceiver Type	SingleWire
NM Message Count	32*
NM Base Address	0x620*
Timing Parameters	
Cycle Time [ms]	10*
Init Delay Time [ms]	70
BusOff Recovery Time [ms]	3700
VNMF Confirmation Time [ms]	500*

Configurable Options	Nm_Gmlan_Gm
User Config File	*
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*

Figure 4-7 Settings for GMLAN

All necessary settings for the VNs are already done derived from the data base file.

Info

Attributes are defined in the data base to specify the characteristics of a network, an ECU, a network node, a message or a signal. These settings are common for all ECUs that are based upon this data base file.
E.g. the attribute **NetworkType** can only be modified in the data base file (dbc). It can be set to **PowerTrain** or **BodyBus** and determines whether it is a CAN Class C or a CAN single wire CAN.

4.2.6 Tp_Iso15765

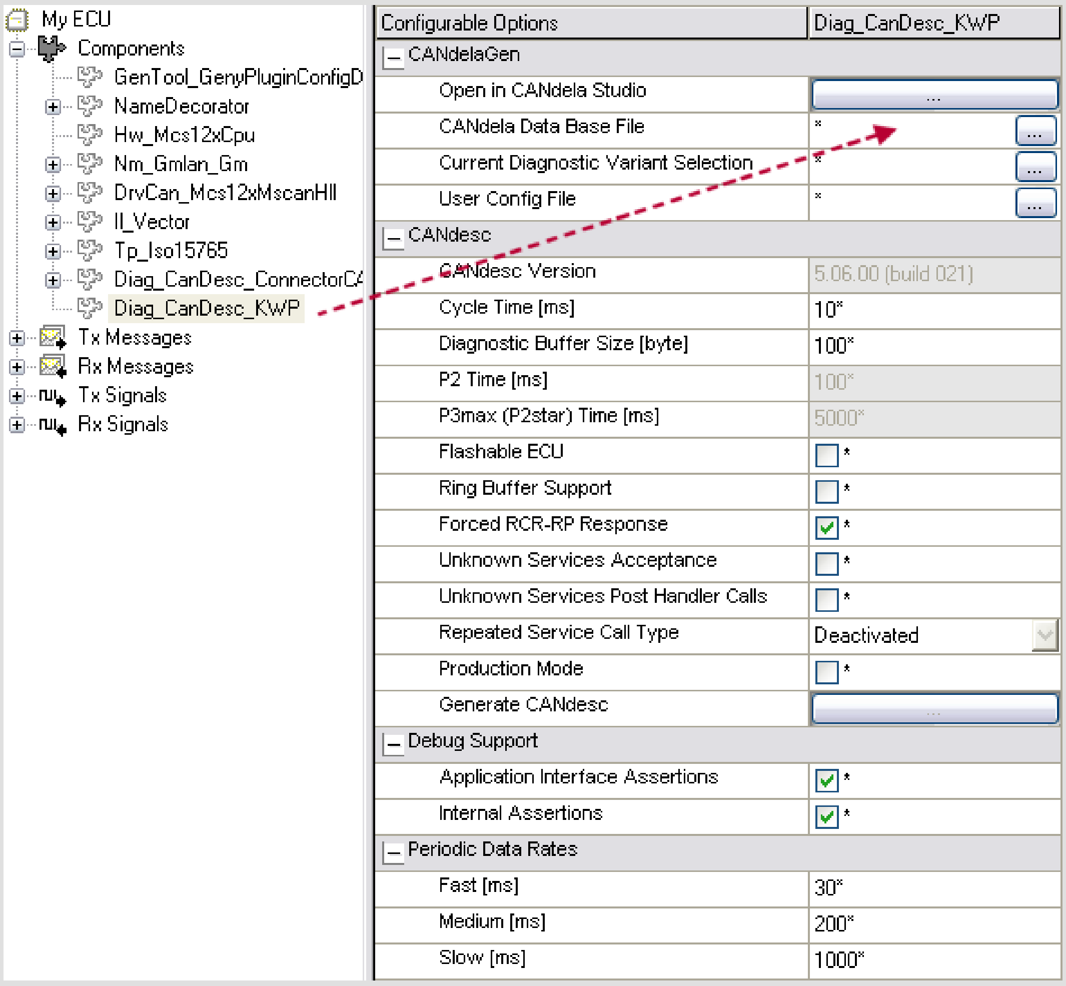
You need the Transport Protocol normally for diagnostic purposes only. Use the default.

Info
For more information about the settings of the transport protocol for your individual needs, refer to the TechnicalReference_TPMC.pdf or the UserManual_TP_GM.pdf.

4.2.7 Diagnostics – Diag_CanDesc_KWP

If you use diagnostic component select it in the **Component Selection** view.

Read-in an appropriate CDD file and use the default settings.



- My ECU
  - Components
    - GenTool_GenyPluginConfigD
    - NameDecorator
    - Hw_Mcs12xCpu
    - Nm_Gmlan_Gm
    - DrvCan_Mcs12xMscanHll
    - Il_Vector
    - Tp_Iso15765
    - Diag_CanDesc_ConnectorCA
    - Diag_CanDesc_KWP
  - Tx Messages
  - Rx Messages
  - Tx Signals
  - Rx Signals

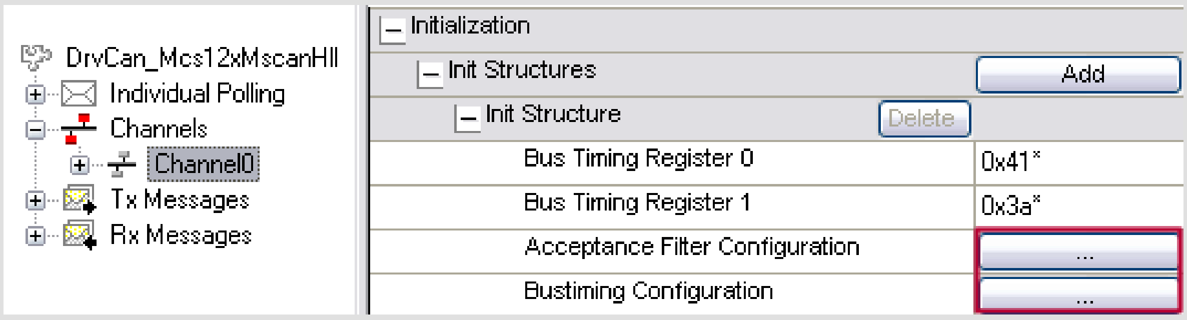
Configurable Options	Diag_CanDesc_KWP
CANdelaGen	
Open in CANdela Studio	...
CANdela Data Base File	* ...
Current Diagnostic Variant Selection	* ...
User Config File	* ...
CANdesc	
CANdesc Version	5.06.00 (build 021)
Cycle Time [ms]	10*
Diagnostic Buffer Size [byte]	100*
P2 Time [ms]	100*
P3max (P2star) Time [ms]	5000*
Flashable ECU	☐ *
Ring Buffer Support	☐ *
Forced RCR-RP Response	✓ *
Unknown Services Acceptance	☐ *
Unknown Services Post Handler Calls	☐ *
Repeated Service Call Type	Deactivated
Production Mode	☐ *
Generate CANdesc	...
Debug Support	
Application Interface Assertions	✓ *
Internal Assertions	✓ *
Periodic Data Rates	
Fast [ms]	30*
Medium [ms]	200*
Slow [ms]	1000*

Figure 4-8 Path to the CDD file on the Configuration View of Diag_CanDesc_KWP

The CDD file has to be created and edited with the Vector Tool CANdelaStudio.

4.2.8 DrvCan_<microcontroller>

There is the access point for two very important setting, the bus timing registers and the acceptance registers.



- DrvCan_Mcs12xMscanHll
  - Individual Polling
  - Channels
    - Channel0
  - Tx Messages
  - Rx Messages

Initialization	
Init Structures	Add
Init Structure	Delete
Bus Timing Register 0	0x41*
Bus Timing Register 1	0x3a*
Acceptance Filter Configuration	...
Bustiming Configuration	...

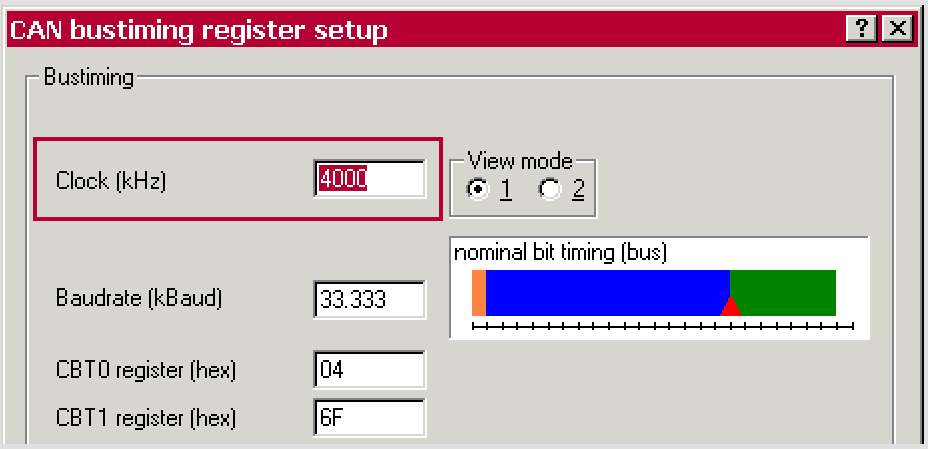
Figure 4-9 Acceptance Filters and Bus timing are channel-dependent settings

As bus timing register and acceptance register settings are channel-dependant, the buttons to open the appropriate windows **[…]** can be found in the configuration view of the channels.

Bus timing registers

The values for the bus timing registers are preset via data base attributes and should be 33.333 kBaud for Single Wire CAN and 500 kBaud for PowerTrain. Check those values again.

Enter the correct value for the Clock [kHz] of your hardware.



Clock (kHz) 4000

Baudrate (kBaud) 33.333

CBT0 register (hex) 04

CBT1 register (hex) 6F

View mode [✓ 1] 2

Figure 4-10 CAN Bus Timing Register Setup

Acceptance registers

Per default the acceptance registers are set to be open. **Optimize filters…** to get a minimum of non-relevant messages into your system.

4.2.9 Settings are Finished

After the settings are done start the generation process of the tool via the yellow flash button.

Use the views **Generated Files** and **Messages** to get information during and after the generation process.

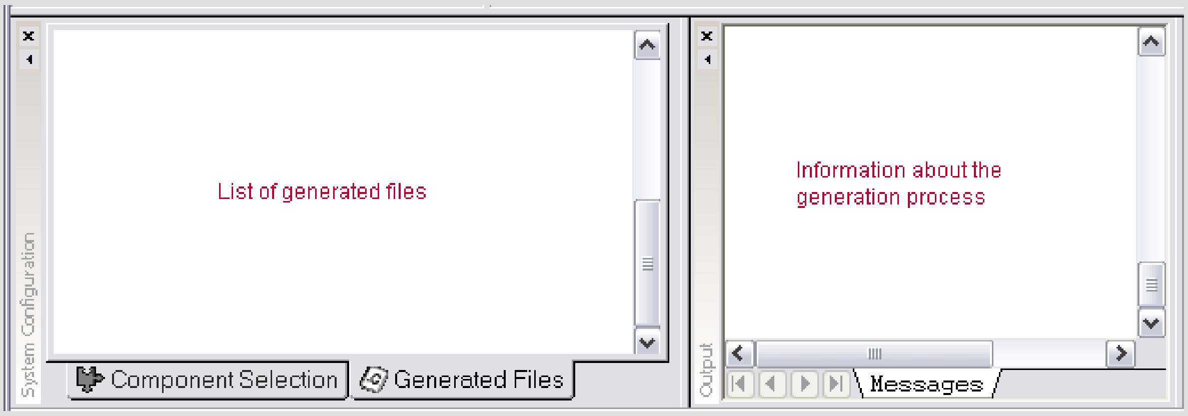


Figure 4-11 Success Window containing Information about Generated Files.

Messages

In the messages view the generation process puts out necessary information like what has been done correctly or which errors occurred.

GENy e.g. checks whether all receive messages of your ECU will pass the acceptance filters. If not, a warning will be displayed, containing the message(s) that won't pass the filter.

Generated Files

This view displays all generated files and the path, where the files have been written to. A good chance to check the correctness of your path settings again.

4.3 STEP 3 Add Files to Your Application

Now you have to add the files of GMLAN Software Components to your application files in your project or makefile.

This includes

- delivered **source code** files and (see the list of files in GM_Readme.pdf)
- **generated** files in the **gendata** folder

4.4 STEP 4 Adaptations For Your Application

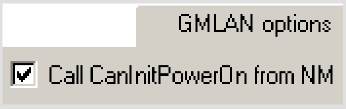
To use the GMLAN Software Components you have to do a few adaptations to your application code.

4.4.1 Includes

The only file you have to include is `il_inc.h`.

```
#include "il_inc.h"
```

4.4.2 Initialization



GMLAN options

☑ Call CanInitPowerOn from NM

```
/* make sure all interrupts are disabled*/
~~CanInitPowerOn();~~ /*If this initialization is
                        demanded to be done from the NM, you do not have to
                        call it.
                        The selection above can be done in the Generation
                        Tool*/

IlInitPowerOn();

TpInitPowerOn();

DescInitPowerOn();  /*optional, if CANdesc is used*/


/*Enable or Disable RX Messages based on Calibrations*/
IlSetRxMessageEnable(<Rx_Message_Name>, 0 or 1 for on or off);


/*Enable or Disable TX Messages based on Calibrations*/
IlSetTxMessageEnable(<Rx_Message_Name>, 0 or 1 for on or off);


IlSetOwnNodeAddress(SWCAN, <srcAddress>);


/*Get learnt source IDs from EEPROM (single channel system)*/
for( index=0; index<kIlNumberOfExtIdRxObjects; index++)
{
```

This is optional and only used to cut down the message set.

For 29-bit IDs only

```
  IlSetRxMessageSourceAddress(index, GetBLearnedSourceId(index));
}
```

4.4.3 Cyclic calls to keep the components running

The CAN Driver is the only component that is event triggered and that does not need a cyclic call to work properly (except for the CAN Driver used in polling mode).

All other components provide a task function that you have to call in the correct call cycle that is entered in the Configuration Tool for each component.

```
/*Network Management task*/
IlNwmTask();

/*Interaction Layer tasks, separated in Tx and Rx*/
IlRxTask();
IlTxTask();

/*Transport Protocol tasks, separated in Tx and Rx*/
TpRxTask();
TpTxTask();

/*Diagnostics task - CANdesc*/
DescTask();
```

4.4.4 Provide Callback functions

The GMLAN Software Components call the application on internal events. For this case there are callback functions to enable the application to know about the program flow and to intervene if necessary. These callback functions must be provided by the application. For the first attempt in most of the cases it is enough to provide an empty callback function.

Callback functions for the Interaction Layer

```
void ApplIlNodeCommActiveFailed( vuint8 srcAddress ){} /*for mixed IDs
                                                        only*/

void ApplIlRxMsgSrcAddressLearned( IlReceiveHandle ilRxHnd, vuint8
srcAddress ){}  /*for mixed IDs only*/
```

```
void ApplIlNodeCommActiveRecovery( vuint8 srcAddress ){}  /*for mixed IDs
                                                           only*/


void ApplIlSourceAddressLearned( vuint8 srcAddress ){}    /*for mixed IDs
                                                          only*/
```

Callback functions for the Diagnostics

```
This depends on the selected diagnostic functions.
```

Callback functions for the Network Management

```
canuint8 ApplNwmSleepConfirmation( NM_CHANNEL_APPLTYPE_ONLY )
{
  return NmSleepOk;
}


void ApplNwmReinitRequest( NM_CHANNEL_APPLTYPE_FIRST  unsigned char VnNr,
unsigned char ReinitRequest ){}


void ApplTrcvrNormalMode( NM_CHANNEL_APPLTYPE_ONLY ){}


void ApplTrcvrSleepMode( NM_CHANNEL_APPLTYPE_ONLY ){}


void ApplTrcvrHighVoltage( NM_CHANNEL_APPLTYPE_ONLY ){}


void ApplNwmVnDeactivated( NM_CHANNEL_APPLTYPE_FIRST  unsigned char VnNr
){}


void ApplNwmVnActivated( NM_CHANNEL_APPLTYPE_FIRST  unsigned char VnNr
){}
```

4.5 STEP 5 Compile And Link

Now we compile and link the project. There should not be any errors or linker problems.

4.6 STEP 6 Test the Software Component

To test the result download the executable to an appropriate target and test it e.g. with CANoe.

5 Further Information

5.1 Full CAN Message Transmission with Extended Ids

A 29bit message in GMLAN is build-up as shown. The lower 8 bits (Source Address) are set by the CAN Driver.

Priority: 3 bits	Parameter ID: 13 bits	Reserved: 5 bits	Source Address: 8 bits

Figure 5-1 Extended CAN Identifier 29bit

If the message is a full CAN message the source address will not be written by the CAN Driver as the full CAN messages are initialized at startup. There are two strategies dealing with full CAN messages:

- Avoid putting extended ID messages to a full CAN object
- Use a pretransmit function to set the source address for every full CAN transmission by hand. The disadvantage is the delay time for calling the function and setting the source address.

Info
It would be sufficient to set the source address only once after initialization of the full CAN transmit objects.

5.2 Take care when working with VNs

Caution
An IlSetEvent for transmitting a corresponding message will be deleted if the ECU is still sleeping.

The transmission of messages caused by an event has to be handled by the application. For this case the application has to perform an `IlPut` to put in the new value of the signal and an `IlSetEvent` to trigger the actual sending of the corresponding message.

A state transition of the Interaction Layer from suspended to running mode will reset the settings of the Interaction Layer including all pending `IlSetEvents`.

A state transition will be performed every time a node wakes up.

Solution

An `IlSetEvent` should only be performed if the Interaction Layer is in running mode. Use the function `IlNwmGetStatus` to check the state of the Interaction Layer.

```
If( IlNwmStateNMActive( IlNwmGetStatus() ) ) == 1
```

```
{
  IlSetEvent…
}
```

Another solution would be to use the **StateOn flags** for signals. They can be activated in the Configuration Tool on the tab **Send signals** and look like e.g.

```
ILGetTxDgnInfStateOn();
```

It returns a true value (0x01) if the corresponding VN is active. Then you could set your event, too.

5.3 Validity of Signals

There are three mechanisms to define a signal to be valid or to be invalid.

- Timeout – message timeout, signal timeout --> flag or function, Configuration Tool
- Validity bit in the same message as the signal
- VDA – Virtual device availability

To find out which signal is using which of the three mechanisms refer to your specification.

Before you use the content of a signal do the testing if the signal is valid right now.

5.4 Signals assigned to different VNs and located in one message

Lets suppose that a message contains signals that are assigned to different VNs. One VN is active the other VN is inactive.

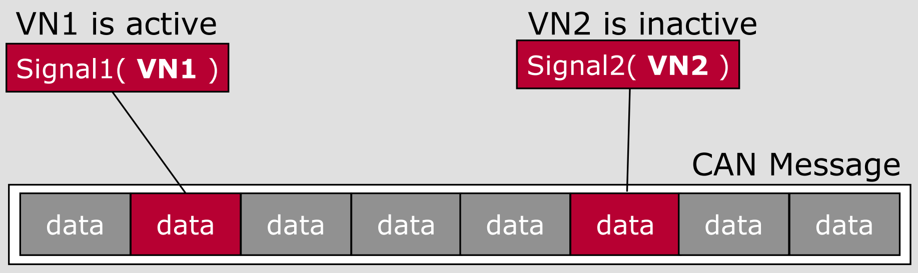


Figure 5-2 Signals of Different VNs in One Message

As a result the confirmation and indication flags of the Signal2 (inactive VN) could be set regardless that the corresponding VN is inactive. Transmission of this signal is possible too and an `IlSetEvent` will provoke the transmission of the message triggered by an inactive VN.

The other ECUs know that the message had been sent by an inactive VN and that the corresponding signals are invalid.

Suggestion
You could check before the `IlSetEvent` if the corresponding VN is active or not.

5.5 Source Learning for Single Wire CAN with Mixed Identifiers (11 bit and 29 bit)

The reason for the source learning mechanism is a flexible monitoring of the receive messages.

Received messages are learned and this information is stored in an array by the Interaction Layer. The application is informed about new learned entries via callback functions
(`ApplIlRxMsgSrcAddressLearned, ApplIlSourceAddressLearned`).

The application only has to store this array in a non-volatile memory and update that array at startup time with the values from the non-volatile memory.

Refer to TechnicalReference_InteractionLayer_GM.pdf in the chapter “Dealing with extended IDs” for more information.

5.6 Activation and deactivation of VNs

The Virtual Networks are activated or deactivated either by the network management (initially active) or by the application. This also depends on the VN configuration.

VN configuration:

- Activator
- Remotely Activated
- Local
- Initially Active

5.6.1 IlNwmActivateVN(channel, VN)

Via this function you activate a virtual network. Only VNs that are configured as activator or local can be manually activated with this function. With it the Interaction Layer is activated to send and receive messages/signals.

5.7 IlNwmDeactivateVN

Deactivates the corresponding virtual network and stops Interaction Layer from transmitting.

6 Index