

Interaction Layer for General Motors

Technical Reference

Il_Vector_Gm with GENy

Version 2.01.02

Authors	Ralf Fritz, Gunnar Meiss, Heiko Hübler
Status	Released

1 Document Information

This document may be revised and appear in several versions. The document will be classified to permit identification of updates and versions.

This user manual is related to the source code version (IL_Vector_Gm) 1.01.00 or higher.

1.1 History

Author	Date	Version	Remarks
Ralf Fritz	2003-08-05	1.00	creation
Ralf Fritz	2004-07-02	1.01	Changed description for ILSetTxMessageEnable
Ralf Fritz	2004-07-16	1.02	Sample corrected. Restriction added.
Ralf Fritz	2005-04-27	1.03	Timeout and Source Learning description extended. New Layout.
Ralf Fritz	2005-08-01	1.04	Adaptation of return values of several functions.
Ralf Fritz	2007-03-29	1.05	Corrected chapter 3.1.2.2 and 3.1.2.3
Ralf Fritz	2007-05-01	1.06	Switch to new documentation template.
Gunnar Meiss	2008-01-17	2.00	Added GENy Support
Heiko Hübler	2012-10-18	2.01.00	Added Robustness Changes Added Clearing Flags on Deactivate VN (ESCAN00061059)
Heiko Hübler	2012-10-26	2.01.01	Changed description of Clearing Flags on Deactivate VN
Heiko Hübler	2013-01-31	2.01.02	Updated GMLAN version (ESCAN00064595) improved the description of Source Address Timeout Supervision (ESCAN00064519)

Table 1-1 History of the Document

1.2 Reference Documents

No.	Source	Title	Version
[1]	Vector	2 2 - (TechnicalReference_CANDriver.pdf).	2.23.00
[2]	Vector	Vector Interaction Layer Technical Reference for GENy. (TechnicalReference_GENy_InteractionLayer.pdf).	2.08.00
[3]	Vector	Tec 2 2 - 2 (TechnicalReference_GMLAN_NM.pdf).	1.07.00
[4]	OSEK/VDX	OSEK/VDX Communication Specification 3.0.3.	3.0.3

Table 1-2 Reference Documents

**Please note**

We have configured the programs in accordance with your specifications in the

questionnaire. Whereas the programs do support other configurations than the one specified in your questionnaire, 2 - release of the programs delivered to your company is expressly restricted to the configuration you have specified in the questionnaire.

Contents

- 1 Document Information2**
 - 1.1 History2
 - 1.2 Reference Documents2

- 2 Component History.....8**
 - 2.1 Il_Vector_Gm Version 1.00.008
 - 2.1.1 What is new?

6.1.1.1	IIInitPowerOn	25
6.1.1.2	IIInit.....	25
6.1.1.3	IIRxTask.....	26
6.1.1.4	IITxTask	26
6.1.1.5	IIRxStateTask	27
6.1.1.6	IITxStateTask.....	28
6.1.1.7	IISetOwnNodeAddress	28
6.2	Service functions	29
6.2.1.1	IISetEvent.....	29
6.2.1.2	IIGetNodeCommActiveState	29
6.2.1.3	IISetRxMessageSourceAddress.....	30
6.2.1.4	IIGetRxMessageSourceAddress	30
6.2.1.5	IISetRxMessageEnable	31
6.2.1.6	IISetTxMessageEnable.....	31
6.2.1.7	IIGetTransmitMessageStatus	32
6.3	Callback functions	32
6.3.1	ApplIISourceAddressLearned.....	33
6.3.2	ApplIIRxMsgSrcAddressLearned	33
6.3.3	ApplIINodeCommActiveRecovery	34
6.3.4	ApplIINodeCommActiveFailed	34
7	Abbreviations.....	36
8	Appendix	37
8.1	Nm_Gmlan_Gm Interface	37
8.1.1	IIRxStart.....	37
8.1.2	IITxStart	37
8.1.3	IIRxStop.....	38
8.1.4	IITxStop	39
8.1.5	IIRxWait	39
8.1.6	IITxWait	40
8.1.7	IIRxRelease.....	40
8.1.8	IITxRelease	41
8.1.9	IIRxActivateVnMsg.....	41
8.1.10	IIRxDeactivateVnMsg	42
8.1.11	IITxActivateVnMsg	43
8.1.12	IITxDeactivateVnMsg.....	43
8.1.13	IIRxStartVnMsgSupervision	44
8.1.14	IIRxDeactivateVnMsgSupervision	45
8.1.15	IIResetRxTimeoutFlags	45
8.1.16	IIRequeueTransmitMessages.....	46

8.2 Interaction Layer Internal Interfaces47

9 Contact.....48

Illustrations

Figure 3-1	Sequence Diagram of Cyclic Transmission	10
Figure 3-2	Sequence Diagram of Event based Transmission.....	11
Figure 3-3	Sequence Diagram of Event based Transmission with Delay	12
Figure 3-4	Sequence Diagram of Cyclic and Event based Transmission in combination.....	13
Figure 4-1	Including Il_Vector_Gm	18
Figure 4-2	Call of the Il_Vector cyclic function.....	20

Tables

Table 1-1	History of the Document.....	2
Table 1-2	Reference Documents.....	2
Table 3-1	Validity Bit Value Interpretation	14
Table 3-2	VDA Bit Value Interpretation	15
Table 3-3	Extended CAN Identifier fields	15

2 Component History

This chapter describes the implementation of the Vector Interaction Layer for General Motors in GENy.

2.1 II_Vector_Gm Version 1.00.00

2.1.1 What is new?

- > The Interaction Layer is configured with GENy.
- > API to handle signal groups.
- > Mask bit support.

2.1.2 What has changed?

- > The Validity bit API is the same API as for Mask bits or signal groups.
- > Reduction of the code size.

2.2 II_Vector_Gm Version 1.01.00

2.2.1 What is new?

- > IIRxDeactivateVnMsg clears now the flags of deactivated messages (see 3.5).

2.2.2 What has changed?

- > The Rx timeout table(IIRxTimeoutTbl) was moved to gmlcal.c and can now be calibrated (post build).

3 Functional Description

3.1 Data Transmission

This chapter describes the data transmission concept of Il_Vector_Gm.



Caution

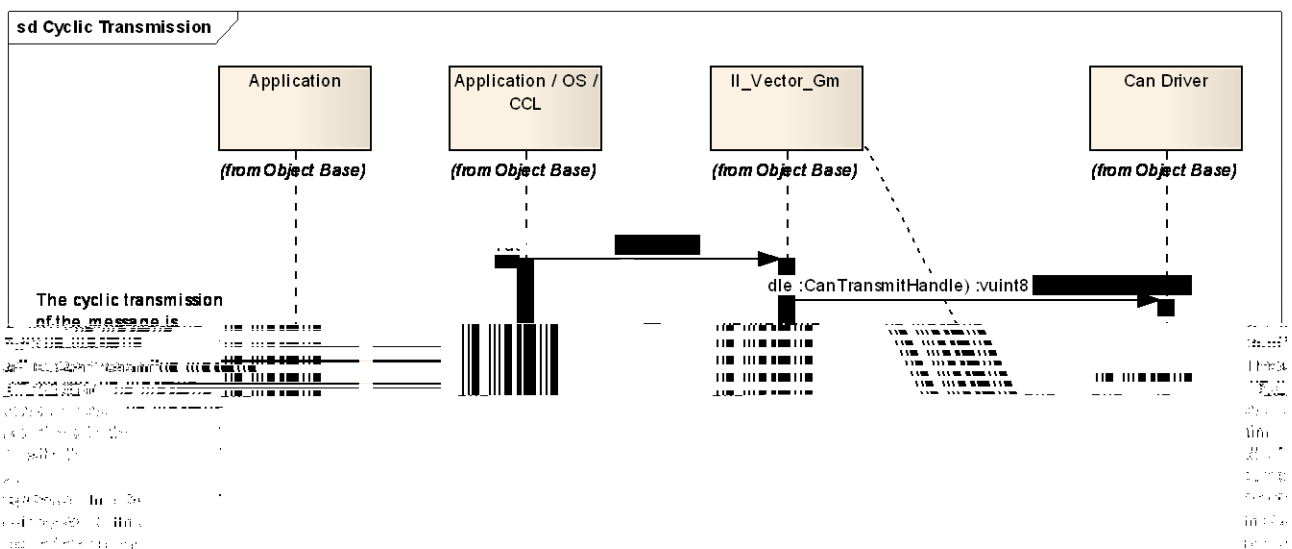
The data transmission differs to the Il_Vector data transmission described in [2].

3.1.1 Cyclic Transmission

The cyclic transmission is configured in the network database with the attributes GenMsgSendType and GenSigSendType (See in chapter 5.1.1 Send Type). If either the message or a signal of the message is configured as cyclic, the message is transmitted periodically. The period of the message is defined with the dbc attribute GenMsgCycleTime (See in [2]).

The cyclic transmission of a message starts automatically, if the Il_Vector_Gm is initialized and the transition II_TxStart is performed for the channel and a VN is active which is related to a signal within a message.

The following sequence diagram describes the cyclic transmission of a message.



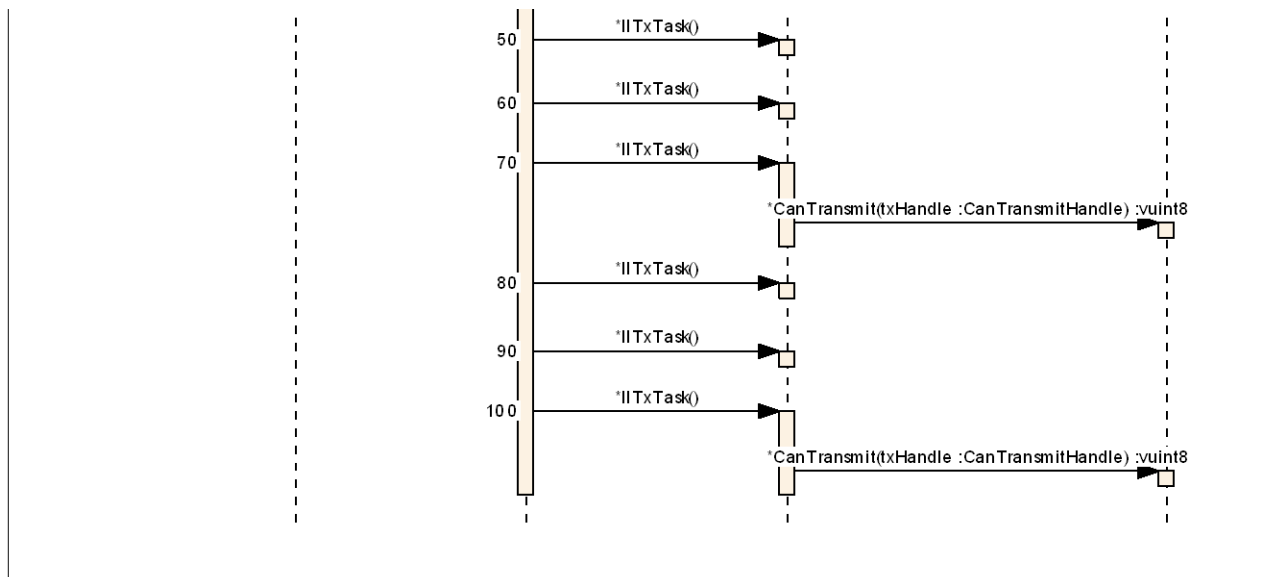


Figure 3-1 Sequence Diagram of Cyclic Transmission

3.1.2 Event Based Transmission

If the GenSigSendType OnAnyChange, OnChangelfActive, OnDelta is defined in the network database for signals, the application has to take care of the transmission event and triggers the transmission of signals.



Caution

If the application does not trigger the transmission, data can get lost.

To implement this functionality, the `II_Vector_Gm` provides to the application `StateOn Flags` per signal and the `IISetEvent` API (See in chapter 6.2.1.1 `IISetEvent`). The following sequence diagram shows the event based transmission in detail. The application checks the VN activity of the signal and if a related VN is active, the application calls `IISetEvent` to set a transmission request. The transmission takes place either within the next call of the `II TxTask` (See Figure 3-2 Sequence Diagram of Event based Transmission), or the transmission is delayed, until the message delay time is elapsed (See Figure 3-3

Sequence Diagram of Event based Transmission with Delay and the description of the GenMsgDelayTime in [21]).



Example

The following code is an example of a event based transmission.

```

/* Write the value to the data buffer for the signal
"EngOilTemp" */
IlPutTxEngOilTemp(5);
/* Check, that the signal is in an active VN */
if (IlGetTxEngOilTempStateOn())
{

```

```

/* Perform the transmission request and
use the generated signal handle from il_par.h as parameter */
IlSetEvent (IlTxSigEngOilTemp);
}

```

The message transmission results of the implemented signal transmission modes. More than one transmission type can be implemented in one message.

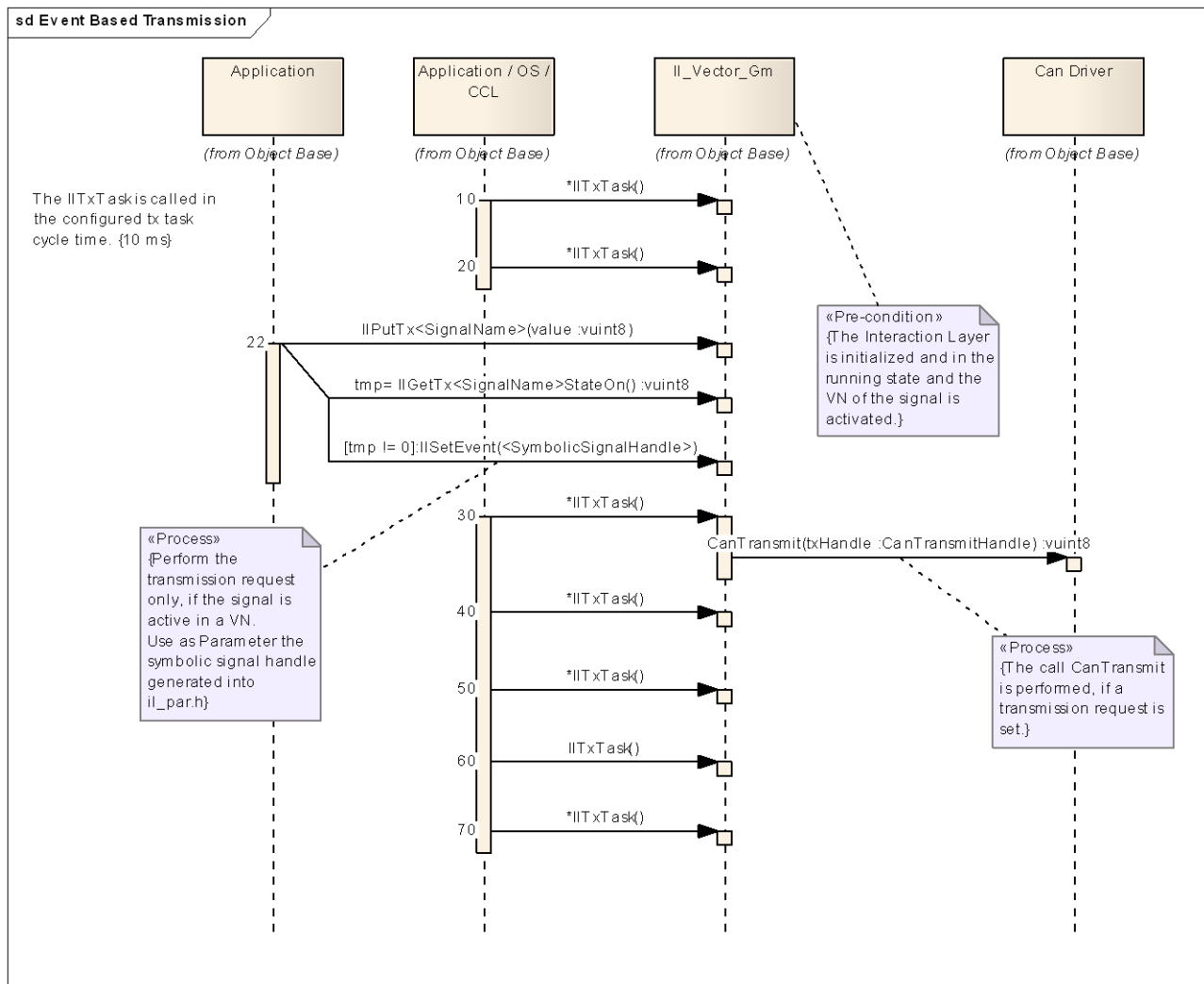


Figure 3-2 Sequence Diagram of Event based Transmission

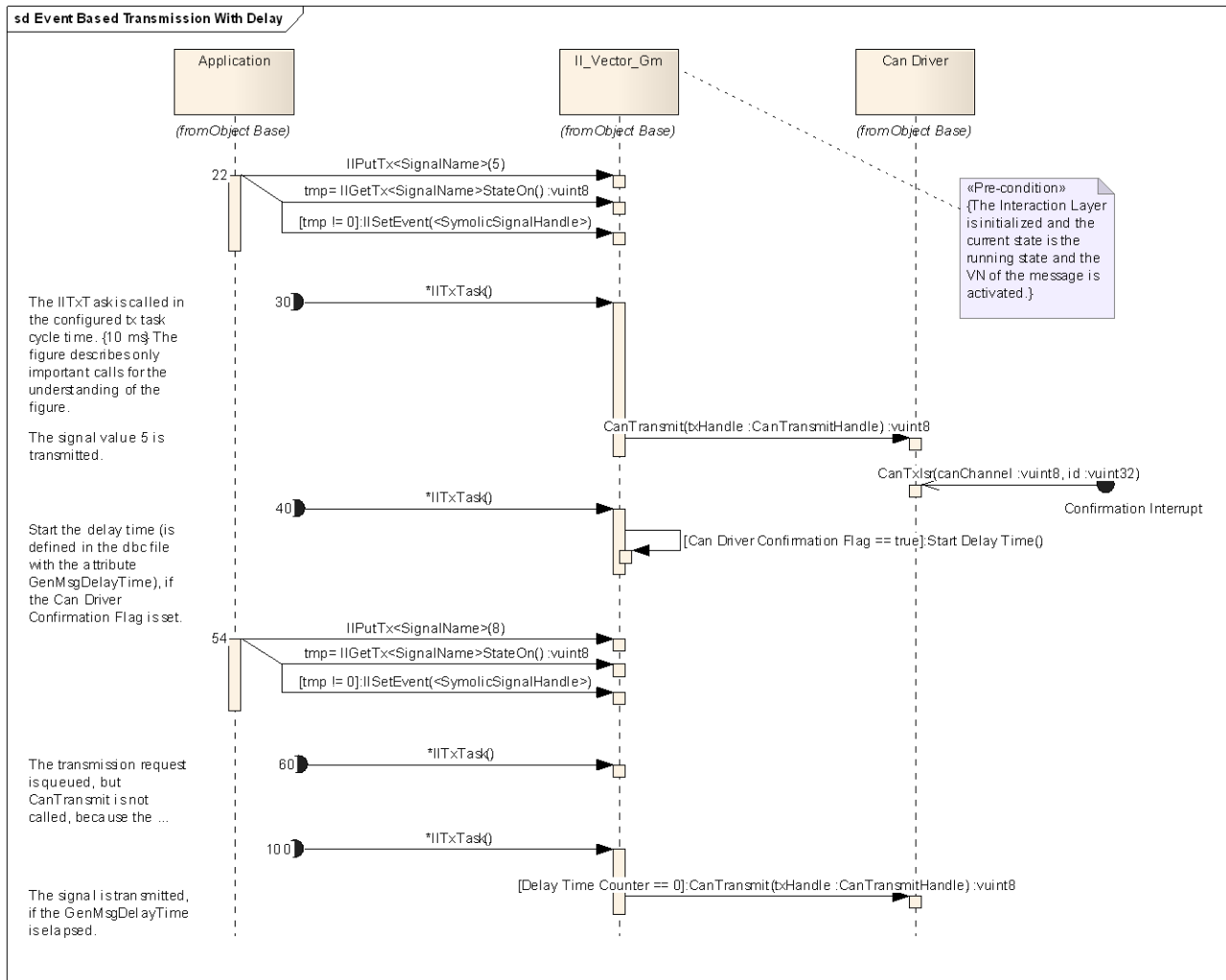


Figure 3-3 Sequence Diagram of Event based Transmission with Delay

3.1.3 Mixed Transmission

The implementation of event based transmission modes for signals can be combined with cyclic transmission. The event based transmission does not influence the periodic transmission event. If the event transmission request is set in the same timeslot as the periodic transmission event, the transmission request is merged.

If the GenSigSendType OnWrite, OnAnyChange, OnChangelfActive, OnDelta is defined in the network database for signals, the application has to take care of the transmission event and triggers the transmission of signals.



Caution

If the application does not trigger the transmission, data can get lost.

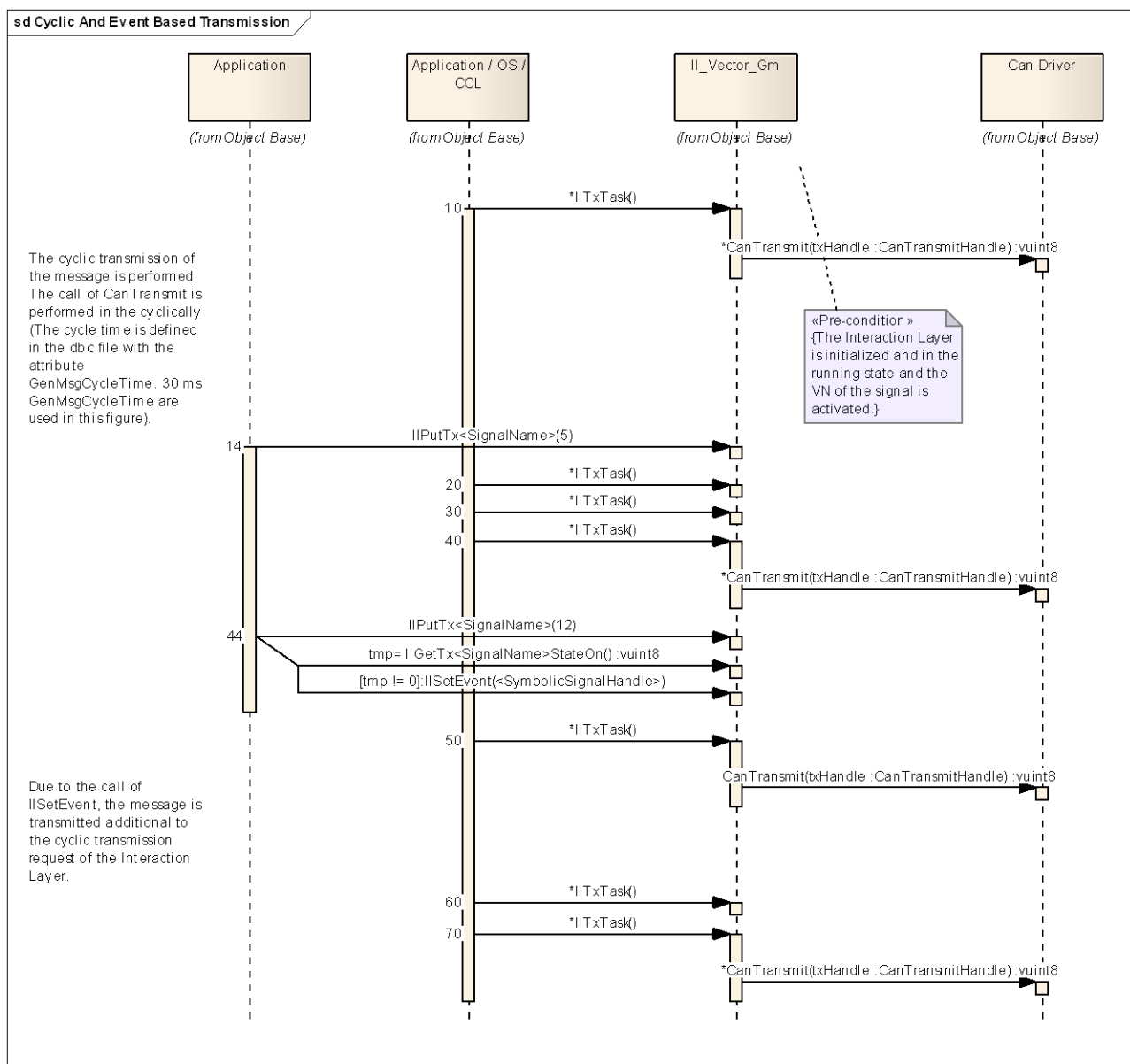


Figure 3-4 Sequence Diagram of Cyclic and Event based Transmission in combination

3.2 Signal Access

The signal API for normal signals is performed as described in [2]. If GENy detects, that the message contains a signal with a related validity, mask or VDA (Virtual Device Availability) bit, the signals are merged into a new created signal group, to access the signals consistently.



Example

EngOilTempV which is grouped into the signal group EngOilTempGroup. A indication flag has been configured EngOilTemp . The shadow buffer for the signal group is provided by the Interaction Layer.

```

vuint8 data = 0;
/* Check, that the signal is received */
if (IlGetRxEngOilTempIndication())
{
    /* Clear the indication flag */
    IlClrRxEngOilTempIndication();
    /* Check, that the signal is in an active VN */
    if (IlGetRxEngOilTempStateOn())
    {
        /* Read the complete signal group to a temporary buffer */
        IlGetRxEngOilTempStateOnGroup();
        /* Check the validity bit */
        if (IlGetRxEngOilTempV() == 0)
        {
            /* Read the signal value and continue data processing */
            data = IlGetRxEngOilTemp();
        }
    }
}

```

Validity Bit Value	Description
0	The signal value is valid.
1	The signal value is NOT valid.

Table 3-1 Validity Bit Value Interpretation

VDA Bit Value	Description
0	The virtual device is NOT available.
1	The virtual device is available.

Table 3-2 VDA Bit Value Interpretation

**Caution**

Due to historical compatibility reasons, the interpretation of validity and VDA bit value is different.

3.3 Extended CAN Identifiers

GMLAN V3.1 is intended to be used with standard and extended CAN identifiers.

This extended CAN Identifiers are introduced to provide the so called Source Learning with Supervision mechanism. This is an extension to the already provided timeout and fault recovery functions.

3.3.1 Source Learning

The 29-bit header of an extended CAN Identifier is separated into different fields:

Priority Field	Bit 28-26	3 bit field used to adjust a message's importance when the transmitter arbitrates for the bus. The value is specified in the network database and cannot be changed at runtime.
Parameter Field	Bit 25-13	13 bit field used to identify the parameter(s) contained within the data field of the message. The parameter(s) will be assigned as ID as they are added to the network database. The value is specified in the network database and cannot be changed at runtime.
Reserved	Bit 12-8	All remaining bits within the 29 bit header shall be reserved for future use. All reserved bits are set to zero by the CAN Driver.
Source Address Field	Bit 7-0	8 bit field used to identify the module which transmitted the message. The source address of an ECU is set by the application in the startup code at runtime via the API <code>IISetOwnNodeAddress</code> (See in chapter 6.1.1.7 <code>IISetOwnNodeAddress</code>).

Table 3-3 Extended CAN Identifier fields

If an extended CAN Identifier is received by an ECU, the priority and source address are filtered by the use of masks generated by the generation tool. The filter for CAN identifier of the CAN controller ignores the parameter values and reserved bits when the message is

received. This enables the system to learn the source address for a specific message. I.e. the source address can be set dynamically.

The application is notified by the call of `AppllSourceAddressLearned` (See in chapter 6.3.1 `AppllSourceAddressLearned`), if a new source address has been learned by the ECU.

If the source address of message changes or the message is received the first time the callback function `AppllRxMsgSrcAddressLearned` (See in chapter 6.3.2 `AppllRxMsgSrcAddressLearned`) is called additional to indicate the relationship between the message and the current ECU, which transmits the message.

No source address is learned at the first ECU start up. The source addresses of all RX messages are set by default to either 254 or 255. A message with a source address of 254 is not mandatory. Messages with a source address of 255 will be mandatory. The source address can be read by the function `llGetRxMessageSourceAddress` (See in chapter 6.2.1.4 `llGetRxMessageSourceAddress`).

If the application needs to identify the current communication state of a learned node the function `llGetNodeCommActiveState` (See in chapter 6.2.1.2 `llGetNodeCommActiveState`) is provided by the Interaction Layer.

During the reception of messages the Interaction Layer learns and stores the source address of each received message. The application has to store the source addresses in a permanent memory location to avoid a new learning phase of the system. If the system is powered up again, the application has to program the previously learned Source addresses via `llSetRxMessageSourceAddress` (See in chapter 6.2.1.3 `llSetRxMessageSourceAddress`). An example of the relearning is provided in chapter 4.2 Initialization.

3.3.2 Source Address Timeout Supervision

The Node Communication Active message (NCA message) is transmitted by each ECU in the network, if the communication is active for a VN (Except VN 0, this one is reserved for diagnosis). With this message it is possible to learn the source address of an ECU, even if no other extended Identifier message is transmitted.

The node timeout supervision starts with the reception of the first extended id message that is no NCA message. If no extended Identifier message with the already learned source address is received, a timeout occurs. The timeout is notified to the application by the call of `AppllNodeCommActiveFailed` (See in chapter 6.3.4 `AppllNodeCommActiveFailed`) with the missing source address. The timeout supervision is performed even for messages, which are not learned by the ECU. If mandatory messages are missing, the source address 255 is passed and 254 for optional messages. If the timeout of a source address is detected, it can be assumed, that all signals related to that source address are in timeout. Additional timeout notifications for extended Identifier messages (message timeout function, signal timeout flags or timeout function) are called or set and timeout default values are set if they are configured.

If an extended Identifier message with the already learned source address is received again (e.g. the NCA message), the application is notified with the call of `AppllNodeCommActiveRecovery` (See in chapter 6.3.3 `AppllNodeCommActiveRecovery`).

3.4 Application Controlled Message Filter

The application can enable and disable the transmission and the reception of messages, which affects all signals included in that messages. See in chapter 6.2.1.5 `IlSetRxMessageEnable` and chapter 6.2.1.6 `IlSetTxMessageEnable`.



Example

Here is an example implementation of the message filter that has been prepared for you.

```
void ApplIlInit(void)
{
    /* Disable the reception of the message RPM_F */
    vError = IlSetRxMessageEnable(IlRxMsgRPM_F,
    kIlMessageDisabled );

    /* Disable the transmission of the message Vspeed */
    vError = IlSetTxMessageEnable(IlTxMsgVSpeed,
    kIlMessageDisabled );
}
```

3.5 Clearing Flags on Deactivate VN

The switch `Enable Clearing Flags on Deactivate VN`, in the GENy GUI, enables clearing flags if the function `IlRxDeactivateVnMsg` is called. This function is called by the NM to deactivate a VN.

These flags are cleared: first value flags, timeout flags, node timeout flags and indication flags.

Only flags of messages which have been deactivated in `IlRxDeactivateVnMsg` are cleared. A message gets deactivated if all VNs a message is in are deactivated.



Info

If you have an old project and activate this feature you may have to adapt your application.



Info

The flags for a signal are cleared when all VNs associated to any signal of the message are deactivated

4 Integration

This chapter includes an example for the integration of the Interaction Layer. Most of the time, we can only describe a single simple configuration just to show how it could look like. We use pseudo code for our

4.1 Include structure

To use the Vector Interaction Layer for GMLAN, only the file `il_inc.h` must be included in all application components that want to use Interaction Layer functionality. The file `can_inc.h` (which provides the CAN Driver interface and data buffers) must not be included separately, it is automatically included by `il_inc.h`.

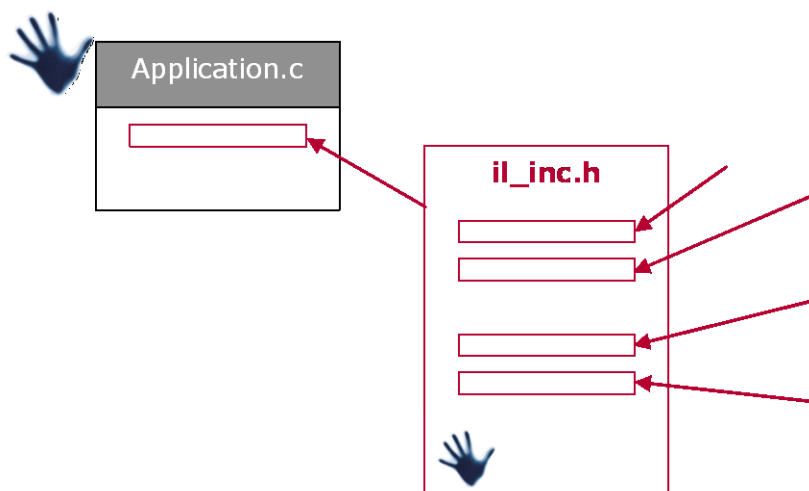


Figure 4-1 Including `Il_Vector_Gm`

4.2 Initialization

If the CCL is not used in the software stack, the application has to initialize the components.



Example

Here is an example, if the initialization has to be implemented by the application.

```
/* Disable interrupts during the initialization of the
Components */
DisableInterrupts();

/* Initialize all components */
CanInitPowerOn();
IlInitPowerOn();
TpInitPowerOn();
```

```
DiagInit();

IlSetOwnNodeAddress(srcAddress);
/* Enabling Interrupts is no longer critical, but not
recommended. */

/* Relearn already learned source addresses from the EPROM
*/
for (Hnd = 0 ; Hnd < iNrOfEPROMELEMENTS ; Hnd++ )
{
    IlSetRxMessageSourceAddress(ReadRxHandleFromEPROM(Hnd),
                                ReadSrcAdressFromEPROM(Hnd));
}

/* Enable interrupts */
EnableInterrupts();
```

4.3 Cyclic function

The `IlRxTask` and `IlTxTask` must be called cyclically as configured in GENy by the Application, OS or CCL.

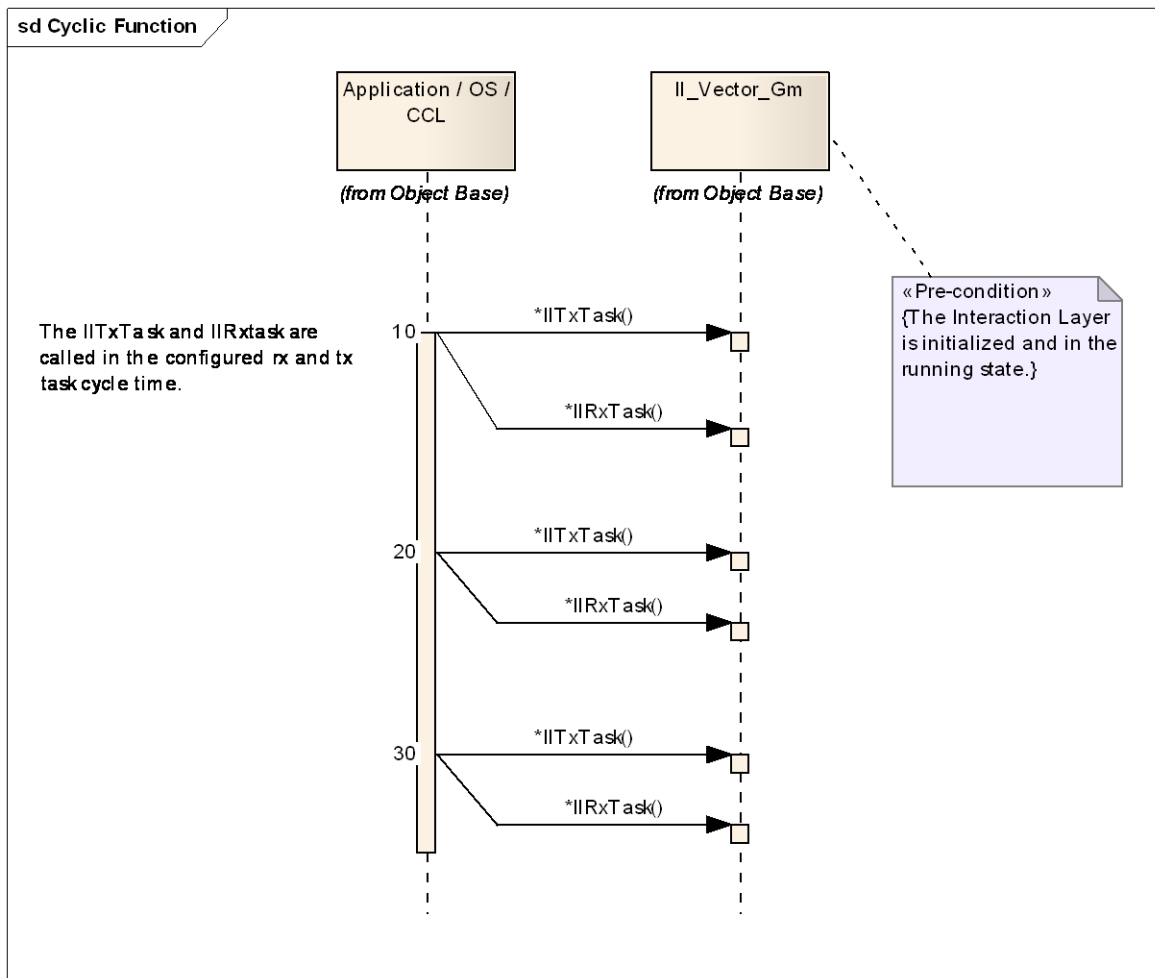


Figure 4-2 Call of the Il_Vector cyclic function



Example

Here is an example, if the task calls have to be implemented by the application.

```

for(;;)
{
    /* periodic call of IlRxTask() and IlTxTask() */
    if (flag_10ms)
    {
        IlRxTask();
        IlTxTask();
        flag_10ms = 0; /* clear flag which was set by a timer
    */
    }
}
    
```

5 Configuration

5.1 Database Attributes

This chapter describes the dbc network database attributes, which can be used with Il_Vector_Gm.



Info

The strings used for the enumerated database attributes are often OEM-specific and can differ here from general descriptions. Do not change the order of string values in enumerated database attributes. The code generator evaluates always the numerical index of the string list.



Caution

2 of enumeration values. Not the value of the attribute is interpreted, the position of the selected value.

5.1.1 Send Type

Name	GenMsgSendType
Description	Message related transmission mode.
Type Of Object	Message
Value Type	Enumeration
Default	CyclicX
Values	CyclicX, SpontanX, NotUsed, NotUsed, NotUsed, NotUsed, NotUsed, NotUsed, NoMsgSendType

Name	GenSigSendType
Description	Signal related transmission mode.
Type Of Object	Signal
Value Type	Enumeration
Default	NoSigSendType
Values	Periodic, NotUsed, NotUsed, NotUsed, NotUsed, NotUsed, NotUsed, NoSigSendType, NotUsed, NotUsed, OnAnyChange, OnChangelfActive, OnDelta

5.1.2 Default Values



Caution

Please note, the attribute GenSigStartValue sets the Default value at initialization time, not if IIRxStart or IITxStart is called. Due to historical and compatibility reasons, this

confusing definition cannot be changed any more.

Name	GenSigStartValue
Description	This value is the default value for the signal, if IllnitPowerOn is called. The string value type can represent hexadecimal and integer values.
Type Of Object	Signal
Value Type	String , Integer*, Float*
Default	0x0
Minimum	0x0
Maximum	0xffffffffffffff

5.1.3 Tx NCA Message

Name	NodeStatusMsgID
Description	This value is the Extended CAN identifier o Ac -- - 2 2 2
Type Of Object	Network
Value Type	Hex
Default	0xFFF800
Minimum	0x1FFFFFFF
Maximum	0xffffffffffffff

Name	NodeStatusMsgCycleTime
Description	- - 2 2 2 2 -- - The message is transmitted, if a virtual network is active.
Type Of Object	Network
Value Type	Integer
Default	1200
Minimum	0
Maximum	65535

Name	NodeStatusMsgTimeoutTime
Description	This value is the timeout time supervision of the Rx Node Communication Active message.
Type Of Object	Network
Value Type	Integer
Default	3000
Minimum	0
Maximum	65535

5.1.4 Timeout Supervision

Name	GenMsgMandatoryToSupervision
Description	This value represents the initial source address, which will be indicated to the application via ApplNodeCommActiveFailed, if no other source address has been learned. If No is set, the source address is set to 254, else 255 is set.
Type Of Object	Message
Value Type	Enumeration
Default	No
Values	No, Yes

Name	GenSigSendOnInit
Description	<p>If a signal of a message has this value set to Handler, the SendOnInit property of the message is activated and preconfigured.</p> <p>The message is transmitted, if IISendOnInitMsg() or IIQueueVnMsg() is called (Called if an initial active VN is activated) or the virtual network is activated (a VN can start locally or remotely if a VNMF message is received).</p>
Type Of Object	Signal
Value Type	Enumeration
Default	NotInitialized
Values	NotInitialized, Application, Handler

Name	GenSigSuprvResp
Description	<p>This value preconfigures the timeout flag and timeout default value.</p> <p>0 : Preconfigure nothing</p> <p>1 : A timeout flag is configured for the signal</p> <p>2 : A timeout default value is configured for the signal</p> <p>3 : A timeout flag and timeout default value is configured for the signal</p>
Type Of Object	Node Mapped Rx Signal
Value Type	Enumeration
Default	None
Values	None, Notify, Substitute, NotifySubstitute

Name	GenSigSuprvRespSubValue
Description	<p>This Value is the timeout default value for the signal, if a timeout occurs.</p> <p>The integer value allows the definition of timeout values for signals with a maximum Length of 4 Bytes.</p>
Type Of Object	Node Mapped Rx Signal
Value Type	Integer
Default	0x0
Minimum	0x0
Maximum	4294967296

6 API Description

The following chapter extends or replaces API function descriptions provided in the Technical Reference of the Interaction Layer [2].

6.1 Administrative functions

6.1.1.1 IInitPowerOn

IInitPowerOn

Prototype	
void IInitPowerOn (void)	
Parameter	
void	none
Return code	
void	none
Functional Description	
This method initializes the <code>II_Vector</code> on all channels. IInit is called for every channel.	
Particularities and Limitations	
The function is called by the Application or Ccl (Communication Control Layer).	
Call context	
The function must be called with disabled interrupts. The function must not interrupt <code>IIRxTask</code> , <code>IIRxStateTask</code> , <code>IITxTask</code> , <code>IITxStateTask</code> , <code>IInit</code> , <code>IIRxStart</code> , <code>IITxStart</code> , <code>IIRxStop</code> , <code>IITxStop</code> .	

6.1.1.2 IInit

IInit

Prototype	
Single Channel	
Single Receive Channel	void IInit (void)
Multi Channel	
Indexed (MRC)	void IInit (CanChannelHandle channel)
Parameter	
channel (Indexed)	Handle of the logical CAN Driver channel.
Return code	
void	none

Functional Description

This method initializes the `II_Vector` on a channel.

Rx and Tx data buffers and flags are set to the initial state. If no default value for a message is defined, the data buffer is set to 0x00. `II_Nwmlnit` of `Nm_Gmlan_Gm` is called if the initialization is performed.

Particularities and Limitations

The function is called by the Application, Ccl (Communication Control Layer) or `IIInitPowerOn`.

Call context

The function must be called with disabled interrupts.

The function must not interrupt `II_RxTask`, `II_RxStateTask`, `II_TxTask`, `II_TxStateTask`, `IIInitPowerOn`, `II_RxStart`, `II_TxStart`, `II_RxStop`, `II_TxStop`.

6.1.1.3 II_RxTask

II_RxTask

Prototype

Single Channel

Single Receive Channel	<code>void II_RxTask (void)</code>
------------------------	------------------------------------

Multi Channel

Indexed (MRC)	<code>void II_RxTask (CanChannelHandle channel)</code>
---------------	--

Parameter

channel (Indexed)	Handle of the logical CAN Driver channel.
-------------------	---

Return code

void	none
------	------

Functional Description

This method must be called periodically in the Rx task cycle time configured in the generation tool. The `II_RxTimerTask` and `II_RxStateTask` are called by this method.

Particularities and Limitations

The function is called by the Application or Ccl (Communication Control Layer).

Call context

The function must be called on task level.

The function must not interrupt `II_RxStateTask`, `II_TxTask`, `II_TxStateTask`, `IIInitPowerOn`, `IIInit`, `II_RxStart`, `II_TxStart`, `II_RxStop`, `II_TxStop`.

6.1.1.4 II_TxTask

II_TxTask

Prototype

Single Channel

Single Receive Channel	<code>void II_TxTask (void)</code>
------------------------	------------------------------------

Multi Channel

Indexed (MRC)	<code>void ILTxTask (CanChannelHandle channel)</code>
Parameter	
channel (Indexed)	Handle of the logical CAN Driver channel.
Return code	
void	none
Functional Description	
This method must be called periodically in the Tx task cycle time configured in the generation tool. The IITxTimerTask and IITxStateTask are called by this method.	
Particularities and Limitations	
The function is called by the Application or Ccl (Communication Control Layer).	
Call context	
The function must be called on task level.	
The function must not interrupt IIRxStateTask, IITxTask, IITxStateTask, IInitPowerOn, IInit, IIRxStart, IITxStart, IIRxStop, IITxStop	

6.1.1.5 IIRxStateTask

IIRxStateTask

Prototype	
Single Channel	
Single Receive Channel	<code>void IIRxStateTask (void)</code>
Multi Channel	
Indexed (MRC)	<code>void IIRxStateTask (CanChannelHandle channel)</code>
Parameter	
channel (Indexed)	Handle of the logical CAN Driver channel.
Return code	
void	none
Functional Description	
This method is called periodically by the IIRxTask. The function can be called in a faster rate than the IIRxTask to check additionally for polled indication events. The usage of the IIRxTask shall be preferred.	
<ul style="list-style-type: none"> - The source addresses queued for the source address learning are learned. - The timeout counter for the source address is started, if the source address is not already known. 	
Particularities and Limitations	
The function is called by the Application or IIRxTask.	
Call context	
The function must be called on task level.	
The function must not interrupt IIRxStateTask, IITxTask, IITxStateTask, IInitPowerOn, IInit, IIRxStart, IITxStart, IIRxStop, IITxStop	

6.1.1.6 IITxStateTask

IITxStateTask

Prototype	
Single Channel	
Single Receive Channel	<code>void IITxStateTask (void)</code>
Multi Channel	
Indexed (MRC)	<code>void IITxStateTask (CanChannelHandle channel)</code>
Parameter	
channel (Indexed)	Handle of the logical CAN Driver channel.
Return code	
void	none
Functional Description	
This method is called periodically by the IITxTask. The function can be called in a faster rate, than the IITxTask, to check additionally for polled confirmation events. The usage of the IITxTask shall be preferred.	
Particularities and Limitations	
The function is called by the Application or IITxTask.	
Call context	
The function must be called on task level.	
The function must not interrupt IIRxStateTask, IITxTask, IITxStateTask, IInitPowerOn, IInit, IIRxStart, IITxStart, IIRxStop, IITxStop	

6.1.1.7 IISetOwnNodeAddress

IISetOwnNodeAddress

Prototype	
Single Channel	
Single Receive Channel	<code>Il_Status IISetOwnNodeAddress (vuint8 srcAddress)</code>
Multi Channel	
Indexed (MRC)	<code>Il_Status IISetOwnNodeAddress (CanChannelHandle channel, vuint8 srcAddress)</code>
Parameter	
channel (Indexed)	Handle of the logical CAN Driver channel.
srcAddress	Source address of this ECU, which will be added to the low byte of each extended ID transmitted by the CAN driver.
Return code	
Il_Status	IL_OK : the node address has been set. IL_ERROR : the node is not configured for the usage of GMLAN extended identifiers and the source address is not accepted.
Functional Description	
This method sets the source address for this ECU.	

Particularities and Limitations

none

Call context

The function must be called on task level after CanInitPowerOn is called AND before interrupts are activated.

6.2 Service functions

6.2.1.1 IISetEvent

IISetEvent

Prototype

```
void IISetEvent (IlTransmitHandle ilTxHnd)
```

Parameter

ilTxHnd	Handle of the Tx message.
---------	---------------------------

Return code

void	none
------	------

Functional Description

This method serves to set a transmission request for a message.

Particularities and Limitations

The function is called by the Application or by IIPutTx method.

Call context

The function can be called on task and interrupt level.

6.2.1.2 IIGetNodeCommActiveState

IIGetNodeCommActiveState

Prototype

Single Channel

Single Receive Channel	<code>vuint8 IIGetNodeCommActiveState (vuint8 srcAddress)</code>
------------------------	---

Multi Channel

Indexed (MRC)	<code>vuint8 IIGetNodeCommActiveState (CanChannelHandle channel, vuint8 srcAddress)</code>
---------------	---

Parameter

channel (Indexed)	Handle of the logical CAN Driver channel.
-------------------	---

srcAddress	Source address of an ECU.
------------	---------------------------

Return code	
uint8	kllNodeUnknown : The given node address has not been learnt. kllNodeFailed : This node is in timeout state. kllNodeActive : Node is sending messages (at least one NCA). kllNodeIdle : Node address is in the list, but is currently inactive.
Functional Description	
This method returns the state of an ECU in the network. The IL searches for the srcAddress and returns the status.	
Particularities and Limitations	
none	
Call context	
The function must be called on task level.	

6.2.1.3 IISetRxMessageSourceAddress

IISetRxMessageSourceAddress

Prototype	
IL_Status IISetRxMessageSourceAddress (ILReceiveHandle ilRxHnd, uint8 srcAddress)	
Parameter	
ilRxHnd	Handle of the Rx message.
srcAddress	Source address of an ECU for this message.
Return code	
IL_Status	IL_OK : the source address is now configured for the Rx message IL_ERROR : ilRxHnd is invalid or ilRxExtIdHnd is inconsistent or channel is inconsistent
Functional Description	
This method sets the source address as learned for a Rx message.	
Particularities and Limitations	
The function is only available if Extended-Identifiers are used.	
Call context	
The function must be called on task level.	

6.2.1.4 IIGetRxMessageSourceAddress

IIGetRxMessageSourceAddress

Prototype	
IL_Status IIGetRxMessageSourceAddress (ILReceiveHandle ilRxHnd, uint8 *pSrcAddress)	

Parameter	
iIRxHnd	Handle of the Rx message.
pSrcAddress	Pointer to where the source address of a message has to be stored.
Return code	
IL_Status	IL_OK : the source address is now returned for the Rx message IL_ERROR : iIRxHnd is invalid or iIRxExtIdHnd is inconsistent or channel is inconsistent
Functional Description	
This method returns the source address a Rx message learnt in the last session.	
Particularities and Limitations	
The function is only available if Extended-Identifiers are used.	
Call context	
The function must be called on task level.	

6.2.1.5 IISetRxMessageEnable

IISetRxMessageEnable

Prototype	
IL_Status IISetRxMessageEnable (ILReceiveHandle iIRxHnd, vuint8 type)	
Parameter	
iIRxHnd	Handle of the Rx message. Use the message handle generated in il_par.h!
type	kIIMsgEnabled or kIIMsgDisabled
Return code	
IL_Status	IL_OK : the Rx messages handle has been activated or deactivated IL_ERROR : the Rx messages handle is out of range
Functional Description	
This method activates or deactivates the reception of a message.	
Particularities and Limitations	
Only messages which are handled by the Interaction Layer can be enabled or disabled. TP and NM messages are not affected (NCA, HLVW, VNMF, USDT and UUDT messages).	
Call context	
The function must be called in the context of ApIIInit().	

6.2.1.6 IISetTxMessageEnable

IISetTxMessageEnable

Prototype	
IL_Status IISetTxMessageEnable (ILTransmitHandle iTxHnd, vuint8 type)	

Parameter	
ilTxHnd	Handle of the Tx message. Use the message handle generated in il_par.h!
type	kllMsgEnabled or kllMsgDisabled
Return code	
IL_Status	IL_OK : the Tx messages handle has been activated or deactivated IL_ERROR : the Tx messages handle is out of range
Functional Description	
This method activates or deactivates the transmission of a message.	
Particularities and Limitations	
Only messages which are handled by the Interaction Layer can be enabled or disabled. TP and NM messages are not affected (NCA, HLVW, VNMF, USDT and UUDT messages).	
Call context	
The function must be called in the context of ApllllInit().	

6.2.1.7 IIGetTransmitMessageStatus

IIGetTransmitMessageStatus

Prototype	
vuint8 IIGetTransmitMessageStatus (IlTransmitHandle ilTxHnd)	
Parameter	
ilTxHnd	Handle of the Tx message. Do not use the generated signal handles for the message indirection!
Return code	
vuint8	(vuint8) 0 : the message is idle and no transmission is expected from the Il kllTxMsgQueued : the transmission is requested to the CAN Driver and wait for confirmation kllTxMsgPending : the message is pending for queuing in the Il.
Functional Description	
The function provides the status of the message.	
Particularities and Limitations	
The function is called by the Application.	
Call context	
The function can be called on task and interrupt level.	

6.3 Callback functions

The following functions have to be implemented by the application if the configuration in GENy activates the callback function.

6.3.1 AppIIISourceAddressLearned

AppIIISourceAddressLearned

Prototype	
Single Channel	
Single Receive Channel	void AppIIISourceAddressLearned (vuint8 srcAddress)
Multi Channel	
Indexed (MRC)	void AppIIISourceAddressLearned (CanChannelHandle channel, vuint8 srcAddress)
Parameter	
channel (Indexed)	Handle of the logical CAN Driver channel.
srcAddress	Source address that has been learned.
Return code	
void	none
Functional Description	
<p>This method is called to indicate, that a new source address had been detected and learned from a node on the network. This method can be influenced by the call of <code>IISetRxMessageSourceAddress</code>. If a node transmits the NCA message and other extended Ids are not used, <code>AppIIIRxMsgSrcAddressLearned</code> will never be called, but this one will be called.</p>	
Particularities and Limitations	
The function is only available if Extended-Identifiers are used.	
Call context	
The function is called in the context the <code>IIRxStateTask</code> or <code>IISetRxMessageSourceAddress</code> .	

6.3.2 AppIIIRxMsgSrcAddressLearned

AppIIIRxMsgSrcAddressLearned

Prototype	
Single Channel	
Single Receive Channel	void AppIIIRxMsgSrcAddressLearned (IIReceiveHandle ilRxHnd, vuint8 srcAddress)
Multi Channel	
Indexed (MRC)	void AppIIIRxMsgSrcAddressLearned (CanChannelHandle channel, IIReceiveHandle ilRxHnd, vuint8 srcAddress)
Parameter	
channel (Indexed)	Handle of the logical CAN Driver channel.
ilRxHnd	Handle of the II Rx message, where the source address has been learnt from.
srcAddress	Source address that has been learned.
Return code	
void	none

Functional Description

This method is called to indicate, that a new source address had been detected and learned from a node on the network from a specific message. This method is not influenced by the call of `ILSetRxMessageSourceAddress`. The function shall be used, to store the learned value and reset it during start-up via `ILSetRxMessageSourceAddress` where the `ILRxHnd` is needed.

Particularities and Limitations

The function is only available if Extended-Identifiers are used.

Call context

The function is called by the `ILRxStateTask`.

6.3.3 AppllNodeCommActiveRecovery

AppllNodeCommActiveRecovery

Prototype

Single Channel

Single Receive Channel	<code>void AppllNodeCommActiveRecovery (vuint8 srcAddress)</code>
------------------------	---

Multi Channel

Indexed (MRC)	<code>void AppllNodeCommActiveRecovery (CanChannelHandle channel, vuint8 srcAddress)</code>
---------------	---

Parameter

channel (Indexed)	Handle of the logical CAN Driver channel.
srcAddress	Source address of an ECU.

Return code

void	none
------	------

Functional Description

This method is called to indicate, that an extended Identifier from the source address has been received again after a communication failure had been detected.

Particularities and Limitations

The function is only available if Extended-Identifiers are used.

Call context

The function is called in the context the `ILRxStateTask` or `ILSetRxMessageSourceAddress`.

6.3.4 AppllNodeCommActiveFailed

AppllNodeCommActiveFailed

Prototype

Single Channel

Single Receive Channel	<code>void AppllNodeCommActiveFailed (vuint8 srcAddress)</code>
------------------------	---

Multi Channel

Indexed (MRC)	void ApplIlNodeCommActiveFailed (CanChannelHandle channel, vuint8 srcAddress)
Parameter	
channel (Indexed)	Handle of the logical CAN Driver channel.
srcAddress	Source address of an ECU.
Return code	
void	none
Functional Description	
This method is called to indicate, that timeout supervision of the Node Communication Active message of a source address has been failed.	
Particularities and Limitations	
The function is only available if Extended-Identifiers are used.	
Call context	
The function is called by the IlRxTimerTask.	

7 Abbreviations

Abbreviation	Description
API	Application Programming Interface
CAN	Controller Area Network
CCL	Communication Control Layer
ECU	Electronic Control Unit
HLVW	High Voltage Wake Up
MRC	Multiple Receive Channel
NCA	Node Communication Active
NM	Network Management
OS	Operating System
USDT	Unacknowledged and Segmented Data Transfer
UUDT	Unacknowledged and Unsegmented Data Transfer
VDA	Virtual Device Availability
VN	Virtual Network
VNMF	Virtual Network Management Frame

8 Appendix

8.1 Nm_Gmlan_Gm Interface

The following methods are interface functions provided by the Interaction Layer for the Gmlan Network Management.

**Caution**

Do not use this functions from within the application unless not explicitly required.

8.1.1 IIRxStart

IIRxStart**Prototype**

Prototype	
Single Channel	
Single Receive Channel	<code>void ILTxStart (void)</code>
Multi Channel	
Indexed (MRC)	<code>void ILTxStart (CanChannelHandle channel)</code>
Parameter	
channel (Indexed)	Handle of the logical CAN Driver channel.
Return code	
void	none
Functional Description	
<p>This method enables the transmission of messages and starts the transmission of periodic messages. The transition "start" of the Tx state machine is performed.</p> <ul style="list-style-type: none"> - The flags used to indicate virtual network activity are cleared. - Requests are queued to be transmitted by the call of <code>ILSendOnInitMsg</code>. 	
Particularities and Limitations	
The function is called by the Application or NM (Network Management).	
Call context	
<p>The function must be called on task level.</p> <p>The function must not interrupt <code>ILRxTask</code>, <code>ILRxStateTask</code>, <code>ILTxTask</code>, <code>ILTxStateTask</code>, <code>ILInitPowerOn</code>, <code>ILInit</code>, <code>ILTxStart</code>, <code>ILRxStop</code>, <code>ILTxStop</code>.</p>	

8.1.3 IIRxStop

IIRxStop

Prototype	
Single Channel	
Single Receive Channel	<code>void IIRxStop (void)</code>
Multi Channel	
Indexed (MRC)	<code>void IIRxStop (CanChannelHandle channel)</code>
Parameter	
channel (Indexed)	Handle of the logical CAN Driver channel.
Return code	
void	none
Functional Description	
<p>This method disables the reception of messages. The transition "stop" of the Rx state machine is performed. The method is used for example to enter the Sleep Mode of an ECU.</p> <ul style="list-style-type: none"> - The timeout flags for the application are cleared. - All Rx virtual networks must be deactivated to call <code>IIRxStop()</code>. 	

Particularities and Limitations

The function is called by the Application or NM (Network Management).

Call context

The function must be called on task level.

The function must not interrupt `II RxTask`, `II RxStateTask`, `II TxTask`, `II TxStateTask`, `II InitPowerOn`, `II Init`, `II TxStart`, `II RxStop`, `II TxStop`.

8.1.4 IITxStop

IITxStop

Prototype

Single Channel

Single Receive Channel	<code>void IITxStop (void)</code>
------------------------	-----------------------------------

Multi Channel

Indexed (MRC)	<code>void IITxStop (CanChannelHandle channel)</code>
---------------	---

Parameter

channel (Indexed)	Handle of the logical CAN Driver channel.
-------------------	---

Return code

void	none
------	------

Functional Description

This method disables the transmission of messages (Sleep Mode). The transition "stop" of the Tx state machine is performed. The method is used for example to enter the Sleep Mode of an ECU.

- All Tx virtual networks must be deactivated to call IITxStop.

Particularities and Limitations

The function is called by the Application or NM (Network Management).

Call context

The function must be called on task level.

The function must not interrupt `II InitPowerOn`, `II Init`, `II RxTask`, `II RxStateTask`, `II RxTimerTask`, `II TxTask`, `II TxStateTask`, `II TxTimerTask`, `II RxStart`, `II TxStart`, `II RxStop`.

8.1.5 IIRxWait

IIRxWait

Prototype

Single Channel

Single Receive Channel	<code>void IIRxWait (void)</code>
------------------------	-----------------------------------

Multi Channel

Indexed (MRC)	<code>void IIRxWait (CanChannelHandle channel)</code>
---------------	---

Parameter

channel (Indexed)	Handle of the logical CAN Driver channel.
-------------------	---

Return code	
void	none
Functional Description	
This method halts the reception of messages. The transition "wait" of the Rx state machine is performed. The method is used for example when the bus-off mode of an ECU was entered.	
Particularities and Limitations	
The function is called by the Application or NM (Network Management).	
Call context	
The function can be called on task and interrupt level.	

8.1.6 IITxWait

IITxWait

Prototype	
Single Channel	
Single Receive Channel	void IITxWait (void)
Multi Channel	
Indexed (MRC)	void IITxWait (CanChannelHandle channel)
Parameter	
channel (Indexed)	Handle of the logical CAN Driver channel.
Return code	
void	none
Functional Description	
This method halts the transmission of messages. The transition "wait" of the Tx state machine is performed. The method is used for example when the bus-off mode of an ECU was entered.	
Particularities and Limitations	
The function is called by the Application or NM (Network Management).	
Call context	
The function can be called on task and interrupt level.	

8.1.7 IIRxRelease

IIRxRelease

Prototype	
Single Channel	
Single Receive Channel	void IIRxRelease (void)
Multi Channel	
Indexed (MRC)	void IIRxRelease (CanChannelHandle channel)

Parameter	
channel (Indexed)	Handle of the logical CAN Driver channel.
Return code	
void	none
Functional Description	
<p>This method restarts the reception of messages from the "Waiting" state. The transition "release" of the Rx state machine is performed.</p> <p>- The timeout counters for all source addresses are restarted.</p>	
Particularities and Limitations	
The function is called by the Application or NM (Network Management).	
Call context	
The function can be called on task and interrupt level.	

8.1.8 IITxRelease

IITxRelease

Prototype	
Single Channel	
Single Receive Channel	<code>void IITxRelease (void)</code>
Multi Channel	
Indexed (MRC)	<code>void IITxRelease (CanChannelHandle channel)</code>
Parameter	
channel (Indexed)	Handle of the logical CAN Driver channel.
Return code	
void	none
Functional Description	
<p>This method resumes the transmission of messages from the "Waiting" state. The transition "release" of the Tx state machine is performed.</p>	
Particularities and Limitations	
The function is called by the Application or NM (Network Management).	
Call context	
The function can be called on task and interrupt level.	

8.1.9 IIRxActivateVnMsg

IIRxActivateVnMsg

Prototype	
Single Channel	

Single Receive Channel	<code>Il_Status IIRxActivateVnMsg (vuint8 ilVnHnd)</code>
Multi Channel	
Indexed (MRC)	<code>Il_Status IIRxActivateVnMsg (CanChannelHandle channel, vuint8 ilVnHnd)</code>
Parameter	
channel (Indexed)	Handle of the logical CAN Driver channel.
ilVnHnd	Il_Vector_Gm internal VN handle.
Return code	
Il_Status	<p><code>IL_ERROR</code> : a parameter check has failed OR the Rx state machine of the channel is not in the running state</p> <p><code>IL_VN_ALREADY_ACTIVE</code> : an already activated VN has been requested for activation</p> <p><code>IL_VN_ACTIVATED</code> : the VN is now activated</p>
Functional Description	
This method starts all Rx messages of a VN.	
Particularities and Limitations	
The function is called by Nm_Gmlan_Gm.	
Call context	
The function must be called on task level and if the Rx state machine of the dependent channel is in the running state.	

8.1.10 IIRxDeactivateVnMsg

IIRxDeactivateVnMsg

Prototype	
Single Channel	
Single Receive Channel	<code>Il_Status IIRxDeactivateVnMsg (vuint8 ilVnHnd)</code>
Multi Channel	
Indexed (MRC)	<code>Il_Status IIRxDeactivateVnMsg (CanChannelHandle channel, vuint8 ilVnHnd)</code>
Parameter	
channel (Indexed)	Handle of the logical CAN Driver channel.
ilVnHnd	Il_Vector_Gm internal VN handle.
Return code	
Il_Status	<p><code>IL_ERROR</code> : a parameter check has failed OR the Rx state machine of the channel is not in the running state</p> <p>an already deactivated VN has been requested for deactivation</p> <p><code>IL_VN_DEACTIVATED</code> : the VN is now deactivated.</p>

Functional Description

This method stops all Rx messages of a VN. If enabled, the flags of the deactivated messages are cleared (see 3.5).

Particularities and Limitations

The function is called by Nm_Gmlan_Gm.

Call context

The function must be called on task level and if the Rx state machine of the dependent channel is in the running state.

8.1.11 IITxActivateVnMsg

IITxActivateVnMsg

Prototype

Single Channel

Single Receive Channel	Il_Status IITxActivateVnMsg (vuint8 ilVnHnd)
------------------------	---

Multi Channel

Indexed (MRC)	Il_Status IITxActivateVnMsg (CanChannelHandle channel, vuint8 ilVnHnd)
---------------	---

Parameter

channel (Indexed)	Handle of the logical CAN Driver channel.
-------------------	---

ilVnHnd	Il_Vector_Gm internal VN handle.
---------	----------------------------------

Return code

Il_Status	<p>IL_ERROR : a parameter check has failed OR the Tx state machine of the channel is not in the running state</p> <p>IL_VN_ALREADY_ACTIVE : an already activated VN has been requested for activation</p> <p>IL_VN_ACTIVATED : the VN is now activated</p>
-----------	--

Functional Description

This method starts all Tx messages of a VN.

Particularities and Limitations

The function is called by Nm_Gmlan_Gm.

Call context

The function must be called on task level and if the Tx state machine of the dependent channel is in the running state.

8.1.12 IITxDeactivateVnMsg

IITxDeactivateVnMsg

Prototype

Single Channel

Single Receive Channel	Il_Status IITxDeactivateVnMsg (vuint8 ilVnHnd)
------------------------	---

Particularities and Limitations

The function is called by Nm_Gmlan_Gm.

Call context

The function must be called on task level and if the Rx state machine of the dependent channel is in the running state.

8.1.14 IIRxDeactivateVnMsgSupervision

IIRxDeactivateVnMsgSupervision

Prototype

Single Channel

Single Receive Channel	<code>Il_Status IIRxDeactivateVnMsgSupervision (vuint8 ilVnHnd)</code>
------------------------	--

Multi Channel

Indexed (MRC)	<code>Il_Status IIRxDeactivateVnMsgSupervision (CanChannelHandle channel, vuint8 ilVnHnd)</code>
---------------	--

Parameter

channel (Indexed)	Handle of the logical CAN Driver channel.
-------------------	---

ilVnHnd	IL_Vector_Gm internal VN handle.
---------	----------------------------------

Return code

Il_Status	<p>IL_ERROR : a parameter check has failed OR the Rx VN is not running OR the Rx state machine of the channel is not in the running state an already supervision deactivated VN has been requested for deactivation</p> <p>IL_VN_DEACTIVATED : the timeout supervision of the VN is now deactivated.</p>
-----------	--

Functional Description

This method stops the Rx timeout supervision for all Rx messages of a VN.

Particularities and Limitations

The function is called by Nm_Gmlan_Gm.

Call context

The function must be called on task level and if the Rx state machine of the dependent channel is in the running state.

8.1.15 IIResetRxTimeoutFlags

IIResetRxTimeoutFlags


Prototype

Single Channel

Single Receive Channel	<code>void IIResetRxTimeoutFlags (void)</code>
------------------------	--

Multi Channel

Indexed (MRC)	<code>void IIResetRxTimeoutFlags (CanChannelHandle channel)</code>
---------------	--

Parameter	
channel (Indexed)	Handle of the logical CAN Driver channel.
Return code	
void	none
Functional Description	
<p>This method clears Rx timeout flags of the Application and internal <code>ilNodeCommActiveTimeoutFlags</code>.</p> <p>- The flags used internal to indicate NCA timeout are cleared.</p>	
Particularities and Limitations	
The function is called by the Application.	
	<p>Caution</p> <p>Do not call this Il internal API from the application!</p>
Call context	
The function can be called on task and interrupt level.	

8.1.16 IlRequeueTransmitMessages

IlRequeueTransmitMessages

Prototype	
Single Channel	
Single Receive Channel	<code>void IlRequeueTransmitMessages (void)</code>
Multi Channel	
Indexed (MRC)	<code>void IlRequeueTransmitMessages (CanChannelHandle channel)</code>
Parameter	
channel (Indexed)	Handle of the logical CAN Driver channel.
Return code	
void	none
Functional Description	
This method queues again all pending Tx messages and set a transmission request for these Tx messages.	
Particularities and Limitations	
<p>The function is called by <code>Nm_Gmlan_Gm</code> during a Busoff to queue again all Tx messages that are pended for transmission to the CAN Driver. The transmit queue is cleared of the CAN Driver id cleared during a bus-off. Therefore all issued messages must be retransmitted by the Il.</p>	
Call context	
The function must be called on task level.	

8.2 Interaction Layer Internal Interfaces

APIs which are not explicitly described in this or any other documentation for the usage shall not be called from the application.

**Caution**

Do not use internal functions from within the application unless not explicitly required.

9 Contact

Visit our website for more information on

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

www.vector-informatik.com