Interaction Layer

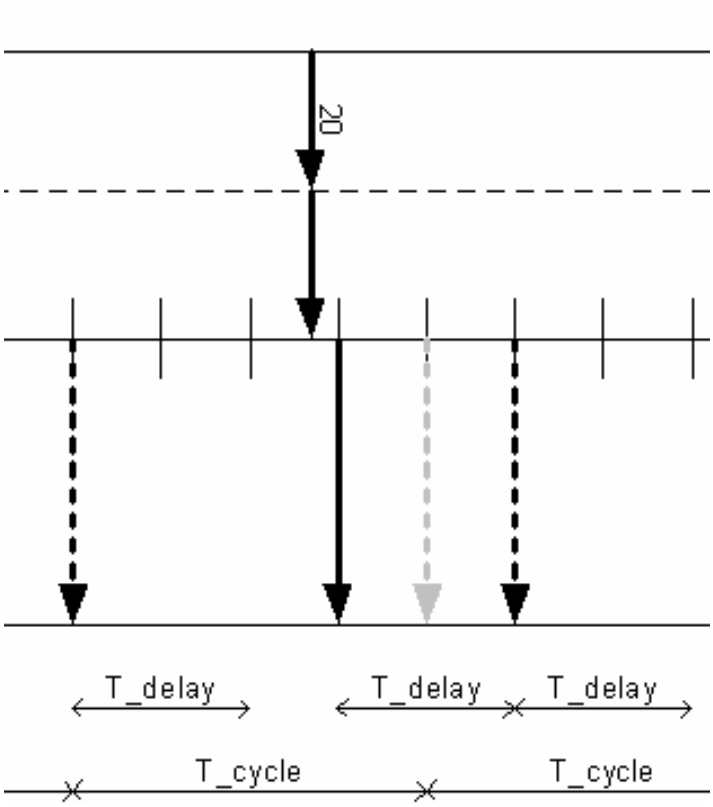
with GENy

User Manual

(Your First Steps)

Version 1.03.01

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:	1.03.01
Status:	released (in preparation/completed/inspected/released)

History

Author	Date	Version	Remarks
Klaus Emmert	2004-04-29	1.00	Converted from Version 0.8 to new User Manual Layout.
Klaus Emmert	2004-05-17	1.1	Usage of vstdlib added (started with IL version 1.83)
Klaus Emmert	2005-06-24	1.02	GENy added as new Configuration Tool.
Gunnar Meiss	2007-05-16	1.03	ESCAN00020395
Gunnar Meiss	2007-07-12	1.03.01	ESCAN00021408 Update Contents

Motivation For This Work

What is a signal?

A Signal is an abstract container for information. It can hold physical values, states or commandos. Signals can concern the complete vehicle or only some control units.

Using the Interaction Layer you do not have to take care about the transmission or reception of signal or the data consistency. If you need the content of a signal, just read it, if a value changed, just write it. All the rest is done by the Interaction Layer.

WARNING

All application code in any of the Vector User Manuals is 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 Welcome to the Interaction Layer User Manual

1.1 Beginners with the Interaction Layer start here?

You need some information about this document? Chapter 2

What is the Interaction Layer? Chapter 3.4

1.2 For Advanced Users

Start reading here. Chapter 4

7 Steps for Interaction Layer integration. Chapter 5

1.3 Special topics

States of the Interaction Layer Chapter 6.1

Generated Names for macros and functions ? Chapter 6.3

Flags and Functions Chapter 0

1.4 Additional Documents dealing with the Interaction Layer

TechnicalReference_InteractionLayer

OEM-specific Documentation

2 About This Document

This document gives you an understanding of the Interaction Layer. You will receive general information, a step-by-step tutorial to get the Interaction Layer running and to use its functionalities.

2.1 How This Documentation Is Set-Up

Chapter	Content
Chapter 1	The welcome page is to navigate in the document. The main parts of the document can be accessed from here via hyperlinks.
Chapter 2	It contains some formal information about this document, an explanation of legends and symbols.
Chapter 3	In this chapter you get a brief introduction in this Interaction Layer and its tasks.
Chapter 4	Here you find some more insight in the Interaction Layer.
Chapter 5	Here are the 7 Steps for you to integrate the Interaction Layer, how to do the necessary settings in the Configuration Tool and how to connect the Interaction Layer with your application.
Chapter 6	This chapter provides you with some further information.
Chapter 7	In this last chapter there is a list of experiences with the Interaction Layer.

2.2 Legend and Explanation of Symbols

You find these symbols at the right side of the document. They indicate special areas in the text. Here is a list of their meaning.

Symbol	Meaning
	The building bricks mark examples.
	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 described Interaction Layer.
	There is something you should take care about.
	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).

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

3 Interaction Layer – An Overall View

One of the important tasks in programming ECUs for modern vehicles is the handling of timers for actions like sending messages or monitoring incoming messages. In some cases this can become quite complex and could be a lot of programming work to do.

3.1 Transmission problems

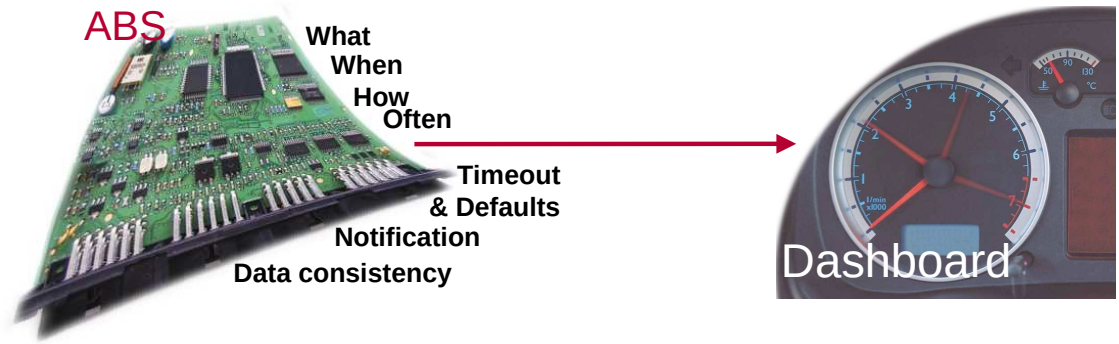


Figure 3-1 Transmission Problems

The names of the macros are derived from the signal names in the database with a pre- and a postfix. You can change the default pre- and suffixes in the Configuration Tool.

The picture above mentions things to remember when transmitting data. You surely know these kinds of problems.

The Interaction Layer handles the classic problems with transmission and reception for you. The information which signal or message should be sent in which timing, which signal triggers a timeout flag when it is not received any more and further information is stored in the data base (DBC file, use CANdb++ for editing DBC data) and realized by the Interaction Layer.

3.1.1 What is left to do for transmission

Regarding the transmission, you just have to fill the signal memory locations (use generated signal/message access macros or functions) with the current values, the transmission itself is done by the Interaction Layer.

Signal access can be a macros or a function. This depends on the signal length, its location in the message and the transmission mode.

The Configuraion Tool always generated the best way to access a signal.

As you access the signals via macros or functions, the data consistency is granted. The Interaction Layer notifies the application in case of important events, such as a successful transmission of a signal (confirmation) or a timeout when a signal has not been received.

Caution
Signal access macros or functions are always **unsigned integer** values !!!

3.2 Reception problems

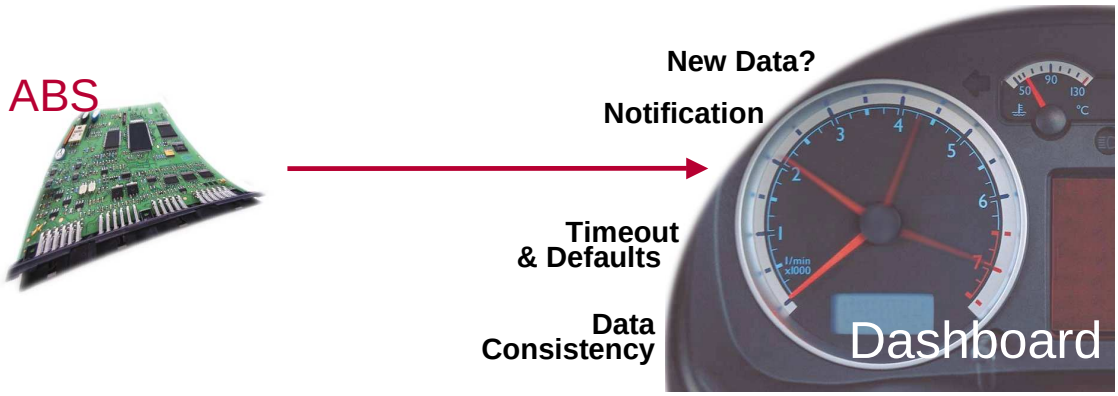


Figure 3-2 Reception Problems

The picture above mentions things to remember when receiving data. You surely know these kinds of problems.

The names of the macros are derived from the signal names in the database with a pre- and a suffix. You can change the default pre- and suffixes in the Configuration Tool tab Names.

3.2.1 What is left to do for Reception

Access the received data via the generated macros or functions any time you need the information. The macros and functions guarantee the data consistency.

Events are signaled to the application via flags (polling) or functions.

You may react on the indications for reception, changed values or timeout. This depends on the demands of your application.

The Interaction Layer will free you from handling the data transmission and reception as far as possible. This job is shifted to the data base engineer who develops the DBC file.

Caution
Signal access macros are always **unsigned integer** values!

3.3 Tools And Files

There is the same DBC-file per bus system (high speed, low speed, etc) for all suppliers to guarantee a common basis for development.

3.3.1 The data base file (DBC file)

The DBC file normally is designed by the vehicle manufacturer and distributed to all suppliers that develop an ECU.

For the Interaction Layer it contains attributes and the information about the transmission modes of the signals (and messages) and e.g. the cycle times. It

contains all information that is necessary for transmitting and receiving signals, which signal should be time monitored etc.

Every supplier uses the SAME DBC file for one type of vehicle to guarantee a common basis for development.

3.3.2 Configuration Tool

The Configuration Tool is a PC-Tool. It is used to configure the CANbedded components to the application's needs. The Configuration Tool generates files that you have to include in your application.

For the Vector Interaction Layer many of the settings are done in the DBC file (see above) and many setting can be done in the Configuration Tool. For each signal flags, default values, access macros or functions and callback functions can be switched to on or off separately.

3.3.3 Generation Process with CANbedded Software Component

The Configuration Tool generates files that contain the configuration and the signal interface of the CANbedded Software Components. In connection with the source code of each component, CANbedded can be compiled and linked (see Figure 3-3).

The standard generation process for Vectors Software Components.

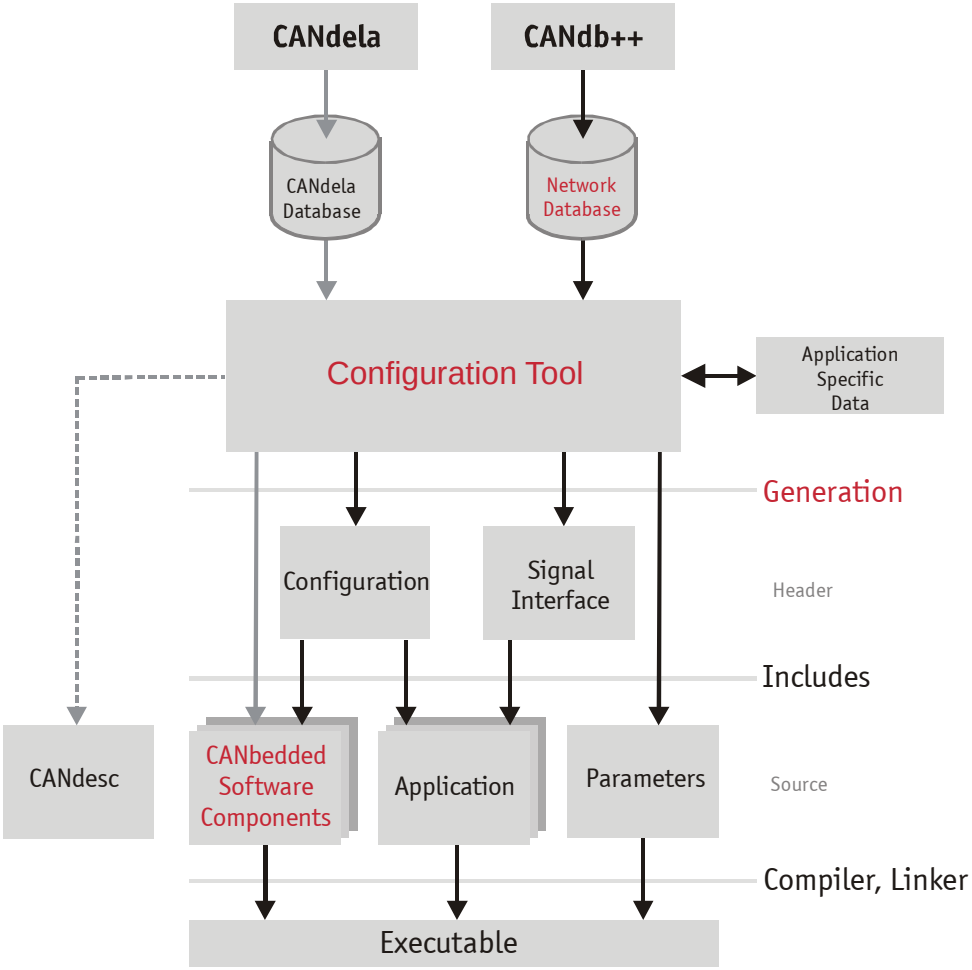


Figure 3-3 Generation Process For Vector CANbedded Software Components

3.4 What Is the Vector Interaction Layer

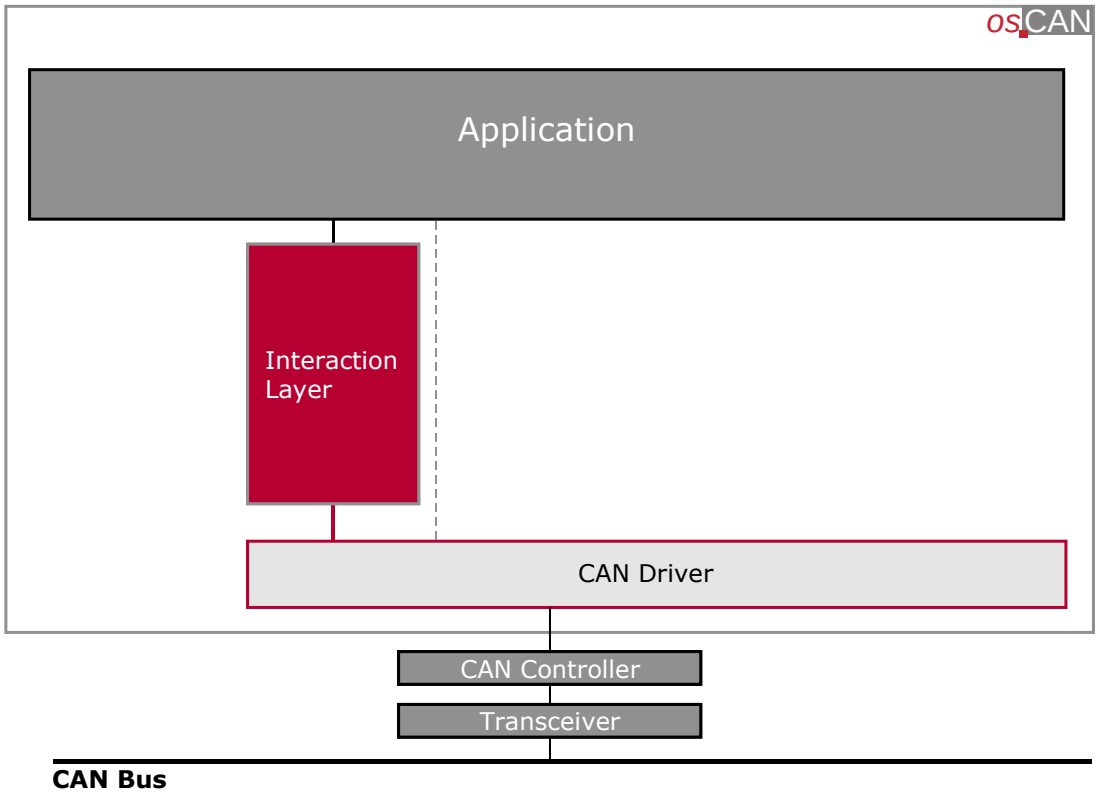


Figure 3-4 Overview CAN Driver, Interaction Layer and Application

As you see the Interaction Layer is a higher layer than the CAN Driver and uses the services of the physical layer for transmission and reception.

3.5 What The Interaction Layer Does

The Vector Interaction Layer is responsible for transmission and reception of CAN messages according their transmission modes, timeout monitoring and setting of default values. It provides a signal interface to the application.

Therefore the programming effort for transmission and reception of signals is reduced on some settings on the DBC and in the Configuration Tool and the basic implementation of the Vector Interaction Layer Component.

4 This Component – A More Detailed View

4.1 Files to form the Interaction Layer

The Interaction Layer consists of 3 sorts of files.

4.1.1 Fix files that form the Interaction Layer

- il.c
- il_def.h

4.1.2 Generated files that must not be changed, too

4.1.2.1 Configuration Tool GENy

- Il_par.c
- Il_par.h
- Il_cfg.h

4.1.3 Configurable files

- il_inc.h or _il_inc.h (the underscore means, this file has to be adapted by you)

4.1.4 il.c

This file contains the code for the Interaction Layer.

You must not change this file at all.

4.1.5 il_def. h

The file il_def.h is the header file of the Interaction Layer.

You must not change this file at all.

4.1.6 Il_par.c

Generated file for the Interaction Layer parameters.

You must not change this file at all.

4.1.7 il_par.h

Generated header for the Interaction Layer parameters.

You must not change this file at all.

4.1.8 il_cfg.h

This is the configuration file of the Interaction Layer. It contains the configuration switches according to your selections in the Configuration Tool.

Do not change this file. You will loose the changes after the next generation process.

4.1.9 il_inc.h

INC stands for include. Here you can add includes you need. Use the delivered standard il_inc.h for this first attempt and look at the default includes as an example.

It is very important that you do NOT change the including order given in this header.

Include this file (as you did it with can_inc.h) in every c file where you need CAN functionality. Il_inc.h “replaces” can_inc.h if you use the Interaction Layer.

4.1.9.1 Vstdlib.c / vstdlib.h

You have to include the Vector standard library beginning with version 1.83 of the Interaction Layer.

4.1.10 Includes when using GENy

To use the Vector Interaction Layer, only the file il_inc.h must be included in all application components that want to use Interaction Layer functionality.

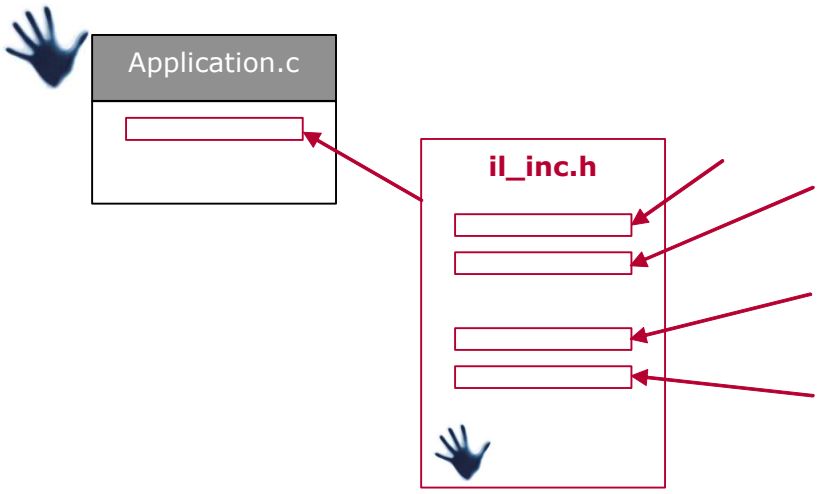


Figure 4-1 Including Vector Interaction Layer

4.2 Handling of the Interaction Layer

The Interaction Layer has to be added to the application and initialized **directly** after the initialization of the CAN Driver. Take care that the interrupts are disabled

Normally the Interrupts are disabled after reset, but this is not granted for any hardware platform.

Make sure the interrupts are disabled.

DO **NOT** USE THE FUNCTION CanInteruptDisable here.

while initialization of the components. Additionally the functions IlRxTask() and IlTxTask() have to be called cyclically within the cycle time adjusted in the Configuration Tool (IL options/Timings IL Vector Channel X).

It is very important for the correct function of the Interaction Layer that the timing you entered in the Configuration Tool and the cycle you call the IlRxTask() and IlTxTask() are the same. This is the basis for the timing of the Vector Interaction Layer.

5 A Basically Running Interaction Layer In 7 Steps

STEP 1 : UNPACK THE DELIVERY
Follow the install shield wizard to unpack the CANbedded Software Components and the Configuration Tool.

STEP 2: CONFIGURATION TOOL AND DBC FILE
Read-in the DBC file in the Configuration Tool and make the configuration settings for the CANbedded Software Components.

STEP 3: GENERATE FILES
Generate the files in the appropriate folders.

STEP 4: ADD FILES TO YOUR APPLICATION
Add the CANbedded C and H files to your project or makefile.

STEP 5: 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 6: COMPILE AND LINK
Compile and link the complete project and download it to your test hardware or development environment.

STEP 7: TEST THE SOFTWARE COMPONENT
Test the software via a suitable test environment.

5.1 STEP 1 Unpack the delivery

The delivery of CANbedded Software Components normally comprises the Configuration Tool and the source code of the software components.

You only have to start the

Setup.exe

and to follow the install shield wizard.

> We recommend creating a shortcut to the Configuration Tool.

The Configuration Tool generates files for your application. It is the connection between your hardware and settings, the requirements of your vehicle manufacturer and the other ECUs, your ECU has to communicate with.

Back to 9 Steps overview

5.2 STEP 2 Configuration Tool and DBC File

We recommend for the integration of the Vector Interaction Layer to have a already running application with already working bus communication (CAN Driver is integrated and running).

Normally the adjustments in the DBC file are done by the data base engineer. To be able to use the Interaction Layer the data base attributes and the corresponding timings have to be set properly.

For the following steps we used a very simple DBC file with only one receive message and two transmit messages.

You can safe much time when you start changes after verifying the functionality of the hardware and software you use as basis for changes.

For more information about how to starup with GENy refer to the GENy online help.

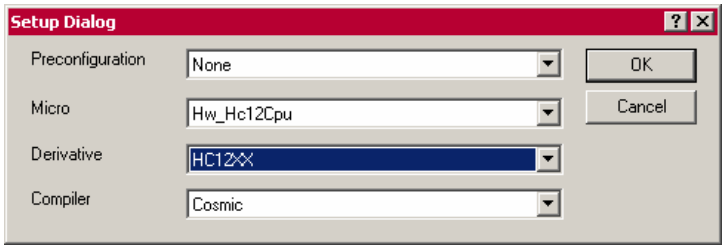
5.2.1 Working with the Configuration Tool GENy

Open the Configuration Tool GENy,

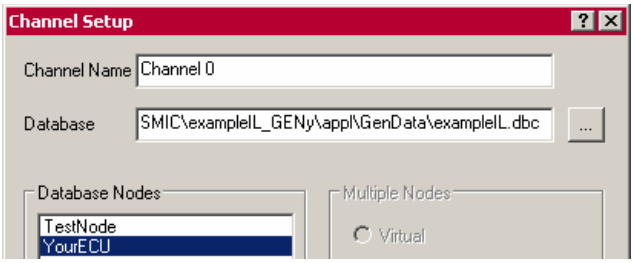
5.2.1.1 Project Setup in GENy

create a new configuration and fill-out the **Setup Dialog**

- Pre-configuration file
- Microcontroller
- Derivative
- Compiler



Select the bus system (here CAN) and fill-out the **Channel Setup** window



Browse for your **Database** file and select your ECU out of the **Database Nodes** list.

Select the **Software Components** Hw_xxx (in this example it is the HC12), the CAN Driver (DrvCan_Hw12MscanIdx) and with Il_Vector the Interaction Layer.

Software Components	My ECU	Channel
Hw_Hc12Cpu	☑	☑
DrvCan_Hc12MscanIdx	☐	☑
Il_Vector	☐	☑

Filetype	Path	Resolved Path
Config Header Files (C)		D:\usr\CodeExamples\COSMIC\exampleIL_GENy\appl\GenData
Parameter Header Files (C)		D:\usr\CodeExamples\COSMIC\exampleIL_GENy\appl\GenData
Parameter Source Files (C)		D:\usr\CodeExamples\COSMIC\exampleIL_GENy\appl\GenData

Set the **Generation Paths** (<Shift> +<F7>) correctly.

5.2.1.2 Interaction Layer Settings in GENy

For this example we want to monitor the reception of the `Signal_2`. If this signal is not received over an adjustable (in the DBC file – use CANdb++ editor) period of time, a timeout occurs.

There are two ways your application can be notified of a timeout,

- via a **flag** or
- a **function**.

In this example we use a function for notification.

Open the **Configuration Options** for **Signal_2** via clicking Signal_2 in the navigation view. Add a timeout function by clicking the **Add** button and entering a name or working with the default as shown.

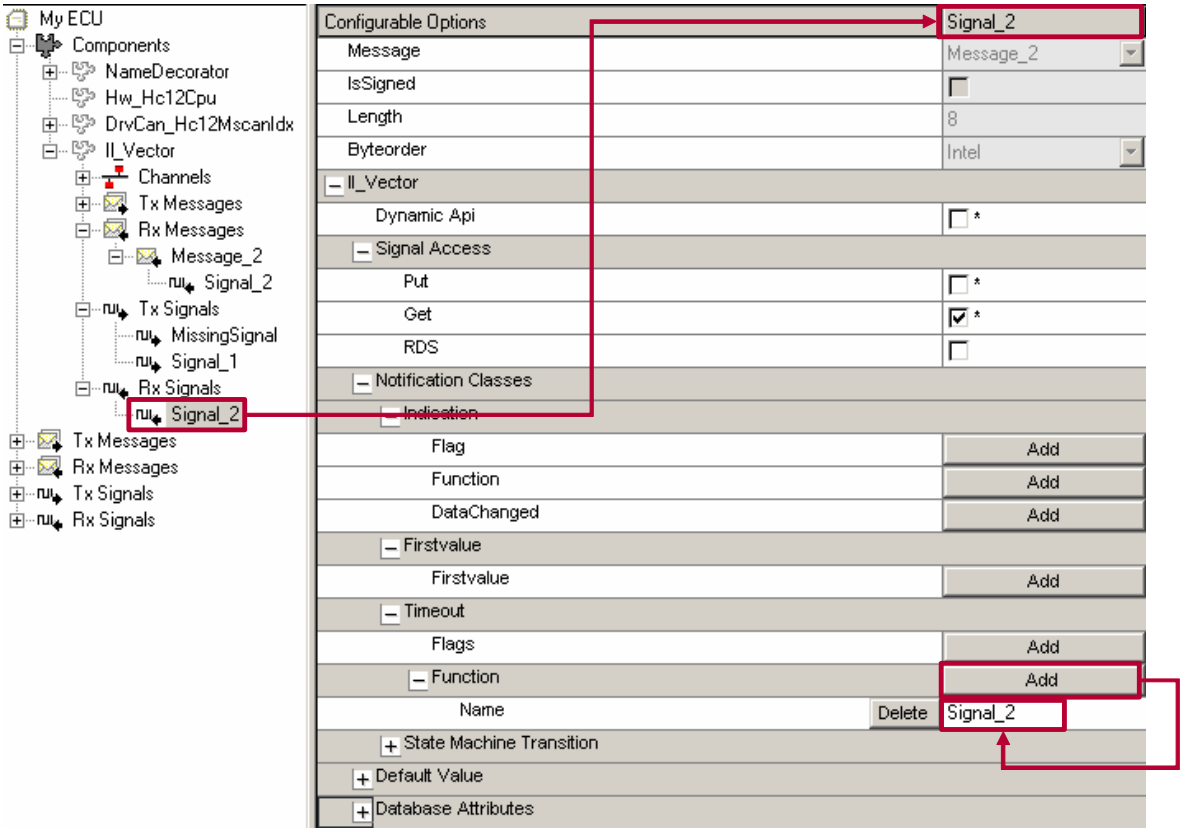


Figure 5-1 Activate the Timeout Monitoring of Signal_2

Back to 9 Steps overview

5.3 STEP 3 Generate Files

Click on the button to start the generation process for GENy.

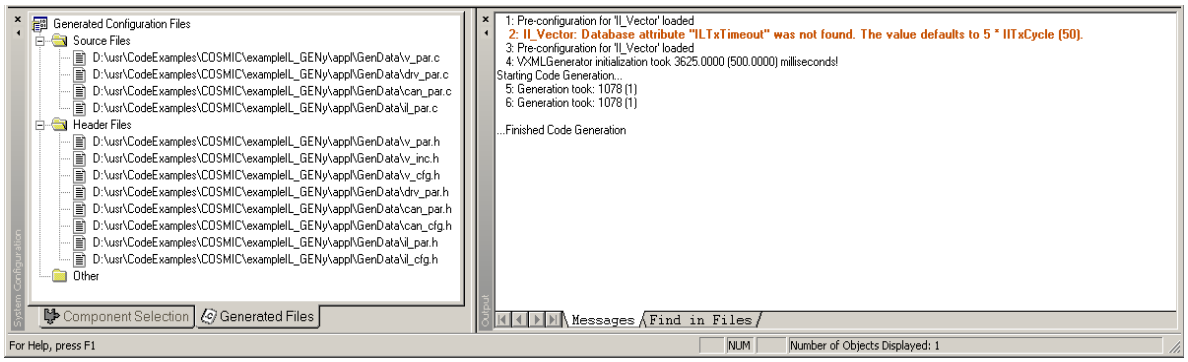


Figure 5-1 Generation Information

The results of the Generation Process are shown in the two views at the bottom of GENy.

Generated Files

This is a list with all file that GENy has generated and the folder information.

Messages

Here the information that occurs during the generation process. Take care of this information after any generation process. Important generation error information could be displayed there as you see in the example above.

Back to 9 Steps overview

5.4 STEP 4 Add Files to Your Application

5.4.1 Using GENy

Copy the core files for the Interaction Layer out of the delivery in the directory you reserved for.

Add those new files together with the generated ones (see Figure 5-1) to the file list of you compiler or makefile.

If you want to apply changes you have done in the Configuration Tool, you must start the generation process again. Remember compiling afterwards.

Starting with IL implementation **version 1.83** the IL uses the Vector standard library **vstdlib** for memory copy and clear routines to avoid calls to the C standard library. Therefore it is required in that case to **link** the file **vstdlib.c** to the project. The v_def.h includes the corresponding header.

Back to 9 Steps overview

5.5 STEP 5 Adaptations For Your Application

Using CANgen and GENy

To be able to compile and link, you have to do a few further adaptations in your application.

This is the example you know from the CAN Driver. For a first test, this simple application without any function except for calling the Interaction Layer Tasks IlRxTask() and IlTxTask() cyclically, is enough.

In the following example application only the modifications are explained and emphasized.

Example for HC12:

```
/* Includes*************************************************************/
#include "can_inc. h"
#include "yourecu. h"
#inlcude”il_inc.h”;    /* with this include, the other two will be
                       included too.*/


unsigned char timerelapsed;   /*the function IlRxTask() and IlTxTask()
                              must not be called in interrupt context.  So
                              this flag is used as an indicator, that a
                              timer interrupt occurred and is polled in
                              the application. */


/*Function prototypes ***************************************************/
void enableInterrupts( void );
void hardwareInit( void );

/*Main Function *********************************************************/
void main(void)
{
/*make sure that the interrupts are disabled or disable interrupts
here.*/
  hardwareInit();

  CanInitPowerOn();
  IlInitPowerOn();/*To use the Interaction Layer you have
                    to do the initialization first.
                    For the case your module does not know
                    this function, use IlInit(); */

timerelapsed = 0; /* timerelapsed = 0x00 →no interrupt occurred
                     timerelapsed = 0xff →   interrupt occurred

  enableInterrupts();
  IlRxStart();
  IlTxStart();
```

It is forbidden to use any CAN functionality before CanInitPowerOn !!!

```
  for(;;)
  {
    /* call of IlRxTask() and IlTxTask() if timerelapsed = 0xff*/
    if( timerelapsed == 0xff)
    {
       timerelapsed = 0;      /*clear the flag to call NmTask()
                               only once after a timer interrupt*/
       IlRxTask();            /*cyclic call for interaction layer*/
       IlTxTask();
    }
  }
}

void ApplCanBusOff( void )
{
  ;
}

void ApplCanWakeUp( void )
{
  /*Call-Back  function  at  the  transition  from  SleepMode  to  sleep
    indication recommended*/
}

void enableInterrupts( void )
{
  CLI();
}


/*This is the function called in case of a timeout of the message_2 that
should be received cyclically*/

void ApplSignal_2SigTimeout( void )
{

  IlPutTxMissingSignal(0x3d);
                              /*Trigger  to  send  this  message  in  case
                             of  a  timeout.  Both  names  are  defined
                             in the YourECU.h.*/
}

void hardwareInit( void )
{
  /*
  Do your hardware specific initializations here.
  Remember your TRANSCEIVER
  */
}

@interrupt void irq_dummy0(void)
{
  for( ;; );   /*all     other     interrupts     except     the     CAN
                          Interrupts are routed to this function.*/
}

/*This is the interrupt function of the timer interrupt.  The timer will
be reset and the timerelapsed indicates the occurrence of an interrupt
for the application (see code above). */
```

```
@interrupt void irq_timer0(void)
{
  /*reload timer*/
  timerelapsed = 0xff;
}
```

You also have to do modifications in the interrupt vector table. The occurrence of a timer interrupt has to lead to the timer interrupt function. In this example you see that the vector for the timer0 interrupt is added

.

Example for HC12:

```
const functptr vectab[] = {        // @0xFFC4 start address of table
    CanTxInterrupt,                // $FFC4    CAN transmit
    CanRxInterrupt,                // $FFC6    CAN receive
    CanErrorInterrupt,             // $FFC8    CAN error
    irq_dummy0,                    // reserved
    irq_dummy0,                    //
    irq_dummy0,                    //
    CanWakeUpInterrupt,            // $FFD0 CAN wake-up
    irq_dummy0,                    //ATD
    irq_dummy0,                    //SCI 2
    irq_dummy0,                    //SPI
    irq_dummy0,                    //SPI
    irq_dummy0,                    //Pulse acc input
    irq_dummy0,                    //Pulse acc overf
    irq_dummy0,                    //Timer overf
    irq_dummy0,                    //Timer channel 7
    irq_dummy0,                    //Timer channel 6
    irq_dummy0,                    //Timer channel 5
    irq_dummy0,                    //Timer channel 4
    irq_dummy0,                    //Timer channel 3
    irq_dummy0,                    //Timer channel 2
    irq_dummy0,                    //Timer channel 1
    irq_timer0,                    //Timer channel 0
    irq_dummy0,                    //Real time
    irq_dummy0,                    //IRQ
    irq_dummy0,                    //XIRQ
    irq_dummy0,                    //SWI
    irq_dummy0,                    //illegal
    irq_dummy0,                    //cop fail
    irq_dummy0,                    //clock fail
    _stext                         //RESET
    };
```

Back to 9 Steps overview

5.6 STEP 6 Compile And Link

Now start your compiler by calling the **makefile** or just clicking the start button, this depends on your development tool chain.

Back to 9 Steps overview

5.7 STEP 7 Test the Software Component

You remember the testing method from the CAN Driver? To test the Interaction Layer we use the same test environment with a few modifications.

5.7.1 Built-up of the test environment

Figure 5-2 The test environment

The network will consist of 2 nodes, a real one (YourECU) and the simulated one, the TestNode (as named in the data base).

Open your CANoe or CANalyzer.

The next step is to **`Associate`** a new **`database`**, the same data base (DBC file) you use for your application.

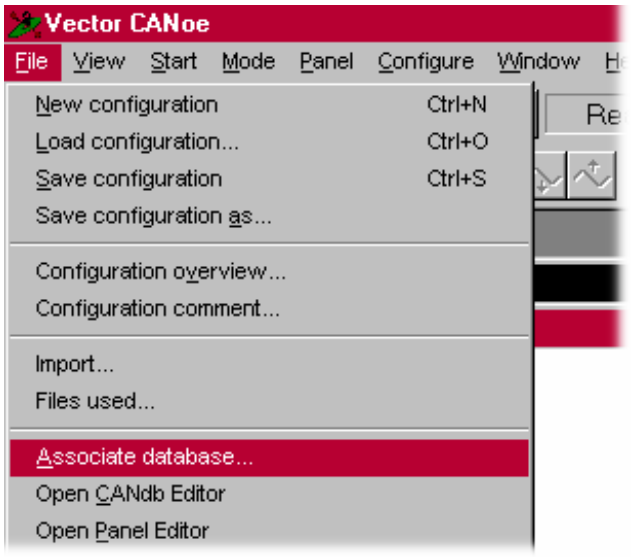


Figure 5-3 How to get an data base into CANoe

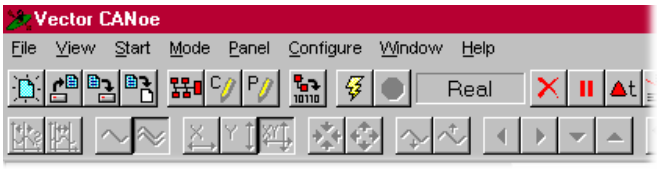


Figure 5-4 Configure menu and Real adjustment

Make sure that the CANoe mode is switched to REAL and you have chosen the same baud rate as in your real node YourECU.

5.7.2 Test of Interaction Layer

Now start your Application on the hardware platform and the CANoe/CANalyzer. Now you should see the cyclic message sent via the Interaction Layer, the `Message_1`. Additionally you see the `Timeout_Message`, because the cyclic message from CANoe is not being sent.

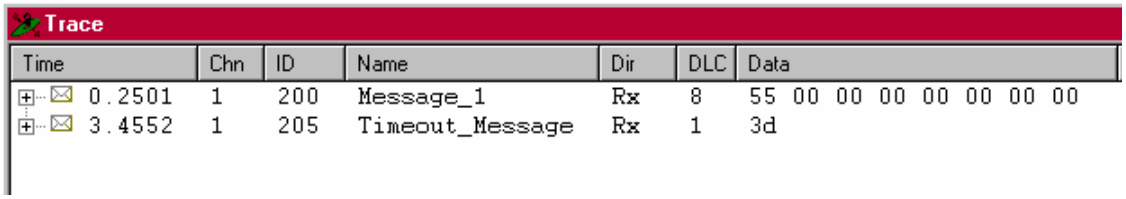


Figure 5-5 A trace of the example application with timeout occurring

!!! CONGRATULATIONS !!!

The Interaction Layer is basically working.

Now try to prevent the timeout from occurring. Insert a generator block and send the message with the ID 201 (Message_2) e.g. in a cycle of e.g. 10ms.

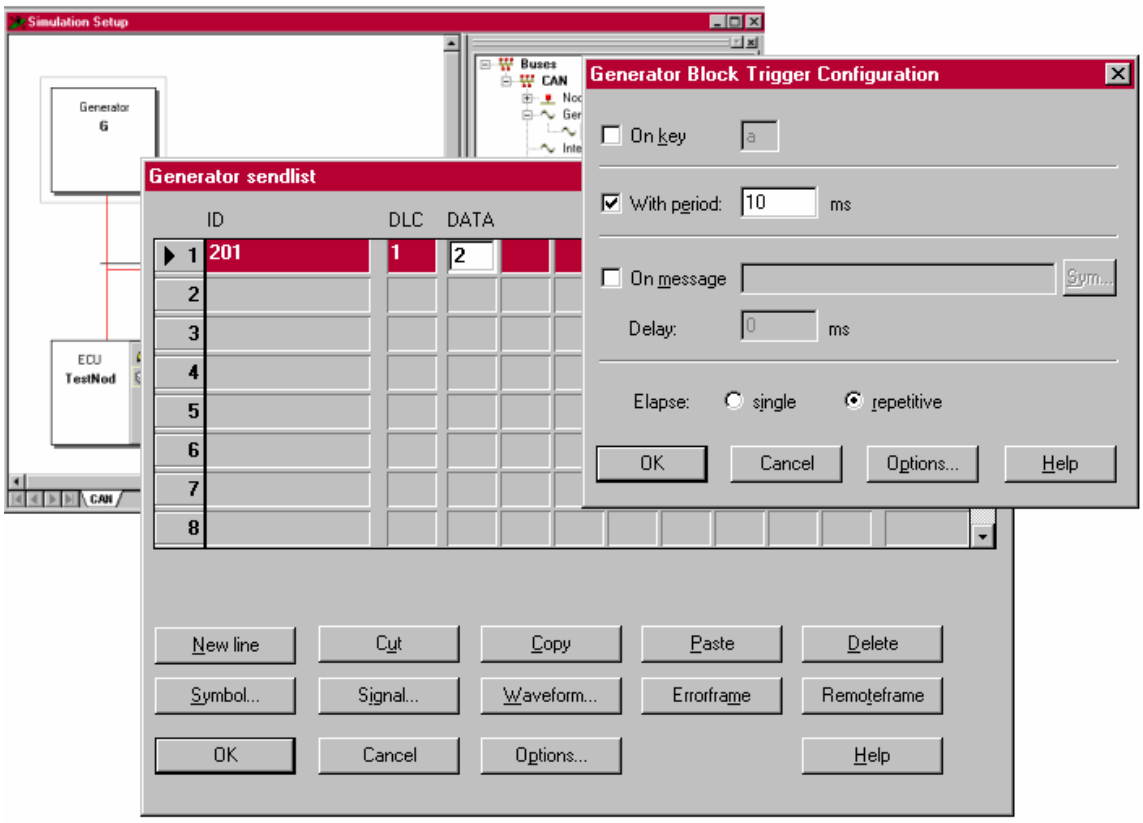


Figure 5-6 Insert a generator block to send the message 201 all 10ms

Restart the application and CANoe and the Timeout_Message will not occur anymore.

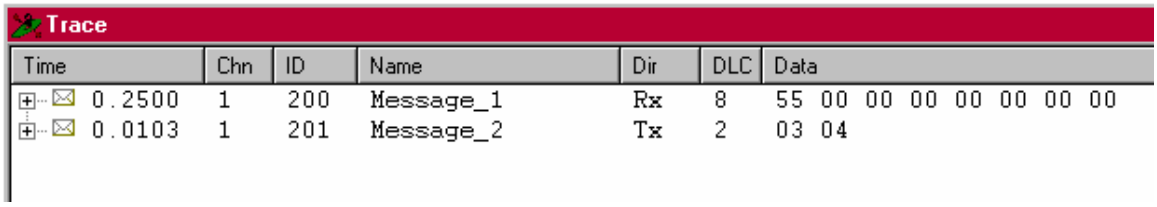


Figure 5-7 A trace without timeout

This is working, too? You have implemented the Interaction Layer correctly for the first time. Now you can go on.

Back to 9 Steps overview

6 Further Information

Now the Interaction Layer is basically working. To get a deeper understanding of this module continue with this chapter. A very detailed description of the API you find in the Technical Reference of the Interaction Layer.

6.1 States of the Interaction Layer

The following figure shows the state machine of the Interaction Layer. There are two identical state machines, one for transmission (**Tx**) and one for reception (**Rx**).

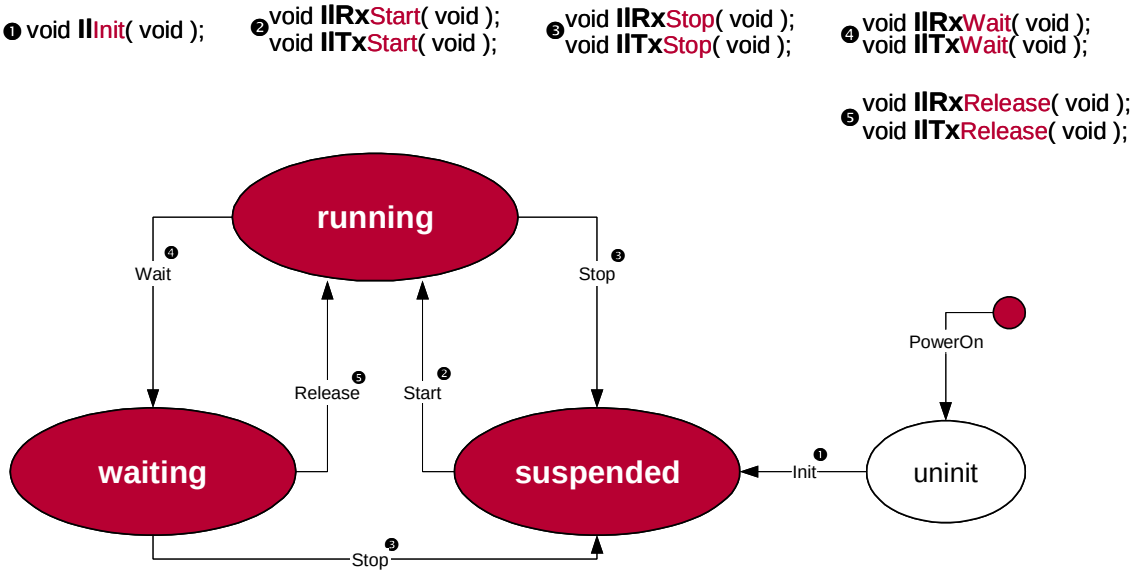


Figure 6-1 The State machine of the Interaction Layer

Use the functions to switch the states of the Interaction Layer to your demands. This becomes very important using the Interaction Layer together with a Network Management.

6.2 Debugging of Interaction Layer

Using the Interaction Layer you can select two main debug options.

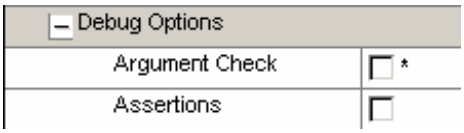


Figure 6-2 Debug options for Interaction Layer

The argument check for the IL functions is used to check the arguments passed to functions of the Interaction Layer. If an error was detected, the return value of the functions will contain an error code. More about this feature see the Technical Reference Interaction Layer.

The assertions you should use only during the development process.

All assertions branch to the function

```
void ApplIlFatalError( ErrorCode );
```

where you can analyse the cause of the assertion.

6.3 Where to get the generated names for the macros and functions

To avoid errors occurring from false writing a recommend to copy and paste the name from the file YourECU.h and ilpar.h.

You know the situation, the files are generated and now you start to develop your application. What is the name of the generated macros for the indication function? What is the correct writing of the callback function? You know the signal name and your pre- and postfixes but:

Using GENy

all this information you get out of the generated file:

`ilpar.h.`

6.4 Usage of flags and functions

The Interaction Layer is signal oriented. So all flags and functions notify events that have to do with signals. There are flags and functions for reception and transmission. See first all flags and functions concerning the reception of data.

Indication Flag **Indication Function**	Indicates the reception of a signal
FirstValue Flag	Can be used to indicate the reception of the signal since the last reset of this flag or of the Interaction Layer
DataChanged Flag	Indicates a changed signal
Timeout Flag **Timeout Function**	Indicates a missing reception of a signal for the adjusted time.

to confirm the transmission there is one flag and one function.

Confirmation Flag **Confirmation Function**	Shows that the message containing the signal has been acknowledged by another node. This acknowledge is sent when at least one ECU has receive the message. The acknowledge triggers a transmit interrupt. Within this interrupt the flag is set and the confirmation function is called.

You can use only the flag, only the function or both combined. This is up to your application.

The confirmation guarantees that the message containing the regarded signal has been sent.

6.5 Data Consistency

A very important part is the knowledge about data consistency and the situations you have to take care for it.

Refer to the chapter dealing with the data consistency in the Technical Reference of the Interaction Layer.

7 Index