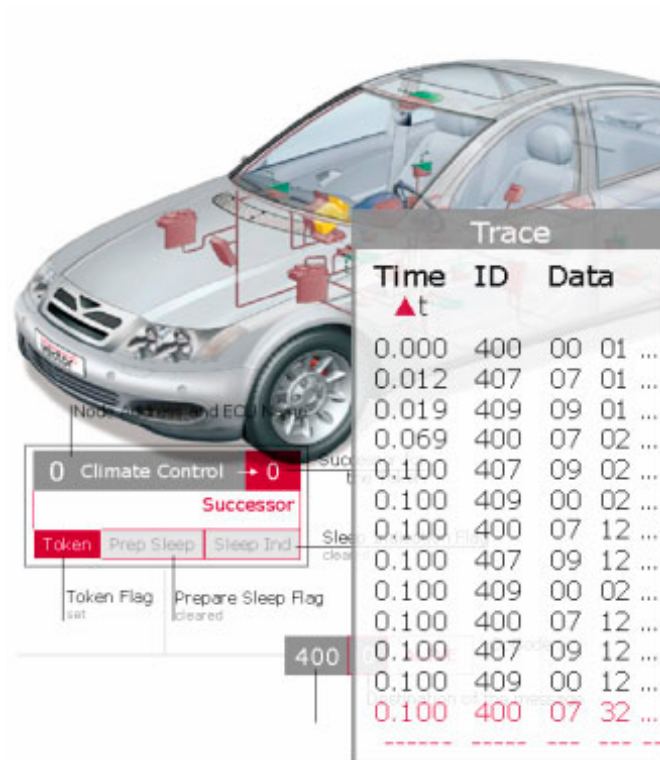


OSEK Network Management

User Manual
(Your First Steps)

Version 1.02



| | |
|-----------------|--|
| Authors: | Klaus Emmert |
| Version: | 1.02 |
| Status: | released (in preparation/completed/inspected/released) |

History

| Author | Date | Version | Remarks |
|--------------|------------|---------|---|
| Klaus Emmert | 2005-06-28 | 1.02 | New Layout, CANGen and new Generation Tool GENy |

Motivation For This Work

Modern vehicles are to be opened very comfortable, via a remote control. How is this functionality realized?

Your signal to open the doors is received via an ECU. This ECU has to wake up the corresponding ECU (e.g. the ECU for the doors and the flashlights) to open the doors.

One technical solution would be a connection between the mentioned ECU just for wake up reasons.

Imagine adding a further ECU to be woken up in this context, the effort of wiring would be enormous. And one main reason for using CAN technology was and is the reduction of wire and connectors, so this cannot be a good solution.

Why should you add connections when all ECUs are already connected via CAN?

What we need is an instance to control the system of ECUs, to recognize whether an ECU is ready to sleep or whether it is not participating in bus communication. This instance has to manage the ECU network.

It is the **Network Management**, described in the following chapters.

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

| | | |
|----------|--|-----------|
| 1 | Welcome to the OSEK Network Managemnt User Manual | 7 |
| 1.1 | Beginners with OSEK Network Management start here ? | 7 |
| 1.2 | For Advanced Users | 7 |
| 1.3 | Documents this one refers to..... | 7 |
| 2 | About This Document | 8 |
| 2.1 | How This Documentation Is Set-Up | 8 |
| 2.2 | Legend and Explanation of Symbols..... | 9 |
| 3 | OSEK Network Management – An Overall View | 10 |
| 3.1 | What Is OSEK Network Management? | 10 |
| 3.2 | What the OSEK Network Management Does | 10 |
| 3.3 | Tools And Files | 11 |
| 3.3.1 | The Network Database | 11 |
| 3.3.2 | Generation Tool | 11 |
| 3.3.3 | Include File Structure | 11 |
| 3.4 | Generation Process with CANbedded Software Component..... | 11 |
| 4 | OSEK Network Management – A More Detailed View..... | 13 |
| 4.1.1 | Files to form the Network Management | 15 |
| 4.1.1.1 | Fix files that form the network management | 15 |
| 4.1.1.2 | Generated files that must not be changed, too | 15 |
| 4.1.1.3 | Configurable files | 16 |
| 4.1.1.4 | can_inc.h..... | 16 |
| 4.2 | Handling of the OSEK Network Management..... | 16 |
| 5 | OSEK Network Management In 7 Steps | 17 |
| 5.1 | STEP 1 Unpack the Delivery..... | 18 |
| 5.2 | STEP 2 Generation Tool and DBC File..... | 19 |
| 5.2.1 | Using CANGen | 19 |
| 5.2.2 | Using GENy | 20 |
| 5.3 | STEP 3 Generate Files | 22 |
| 5.3.1 | Using CANGen Generation Tool..... | 22 |
| 5.3.2 | Using new Generation Tool..... | 22 |
| 5.4 | STEP 4 Add Files to your Application | 24 |
| 5.5 | STEP 5 Adapt Your Application Files..... | 25 |
| 5.6 | STEP 6 Compile And Link Your Project..... | 28 |
| 5.7 | STEP 7 Test It Via CANoe | 29 |

| | | |
|----------|---|-----------|
| 6 | Further Information | 32 |
| 6.1.1 | PowerOff / On | 32 |
| 6.1.2 | Normal..... | 33 |
| 6.1.3 | Sleep Indication..... | 33 |
| 6.1.4 | WaitBusSleep..... | 33 |
| 6.1.5 | Sleep | 33 |
| 6.1.6 | NmOsekInit(NM_SLEEPIND/NM_CANSLEEP/NM_NORMAL)..... | 34 |
| 6.1.7 | GotoMode(Awake/BusSleep)..... | 34 |
| 6.1.8 | ApplNmCanNormal | 34 |
| 6.1.9 | ApplNmCanSleep..... | 35 |
| 6.1.10 | ApplNmCanBusSleep | 36 |
| 6.1.11 | ApplCanWakeUp..... | 36 |
| 6.1.12 | Extended Callbacks | 37 |
| 6.1.13 | ApplNmWaitBusSleepCancel () | 37 |
| 6.1.14 | ApplNmWaitBusSleep () | 38 |
| 6.1.15 | ApplNmBusStart () | 38 |
| 6.1.16 | Handling of the Bus Transceivers | 38 |
| 7 | List Of Experiences | 39 |
| 7.1 | Topic 1 | 39 |
| 8 | Index | 1 |

Illustrations

| | | |
|-------------|---|----|
| Figure 3-1 | Overview CAN Driver, Network Management and Application | 10 |
| Figure 3-2 | Include Files | 11 |
| Figure 3-3 | Generation Process For Vector CANbedded Software Components | 12 |
| Figure 4-1 | The ECUs form a ring via the token ring principle..... | 13 |
| Figure 4-2 | Broadcast Mechanism..... | 14 |
| Figure 4-3 | Example of contents of a Network Management Message. This is dependent on the vehicle manufacturer..... | 14 |
| Figure 5-1 | Settings for Network Management in the Generation Tool | 19 |
| Figure 5-2 | Setup Dialog Window and Channel Setup Window to Create a New Configuration..... | 20 |
| Figure 5-5 | Generation Process | 22 |
| Figure 5-6 | Information About the Generated Files and the Generation Process | 22 |
| Figure 5-7 | Include Components Into Your Application | 24 |
| Figure 5-8 | Test environment | 29 |
| Figure 5-9 | Add the database to CANoe | 29 |
| Figure 5-10 | Insert a new network node | 30 |
| Figure 5-11 | Node Configuration | 30 |
| Figure 5-12 | Save the node without any code | 31 |
| Figure 5-13 | Compile all nodes..... | 31 |
| Figure 6-1 | The states of the OSEK Network Management | 32 |
| Figure 6-2 | The transitions between the state of the Network Management..... | 34 |

1 Welcome to the OSEK Network Management User Manual

1.1 Beginners with OSEK Network Management start here ?

You need some **information** about this document?

What is **OSEK Network Management**?

Chapter 2

Chapter 3.1

1.2 For Advanced Users

Start reading **here**.

7 Steps for Flash Bootloader integration.

1.3 Documents this one refers to...

TechnicalReference_OSEKNetworkManagement.pdf

2 About This Document

This manual gives you an understanding of the OSEK Network Management. You will receive general information and a step-by-step tutorial how to include and use the functionalities of the Network Management.









For more detailed information about the OSEK Network Management and its API refer to the Technical Reference
(TechnicalReference_OSEKNetworkManagement.pdf).

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 the OSEK Network Management and its tasks. |
| Chapter 4 | Here you find some more insight in the OSEK Network Management. |
| Chapter 5 | Here are the 7 steps for you to integrate the OSEK Network Management, how to do the necessary settings in the Generation Tool and how to connect these settings with your application. |
| Chapter 6 | This chapter provides you with some further information, e.g. information about the Network Management state machine. |
| Chapter 7 | In this last chapter there is a list of experiences. |

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 OSEK Network Management. |
|  | 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 OSEK Network Management – An Overall View

A modern vehicle contains an increasing amount of ECUs and bus systems (powertrain (high speed CAN), body CAN (low speed CAN), MOST, LIN, etc). All ECUs within the same bus system have to communicate with the same “language”. ECUs at the border to other bus systems have to translate the information from one “language” into another, they are called gateway.

ECUs within one bus system use the same physical communication layer, e.g. CAN or MOST. The last step to make the communication possible, is the usage of a common “language”.

In the area of CAN communication, this “language” is determined by the dbc file.

3.1 What Is OSEK Network Management?

The Figure 3-1 illustrates the relationship between the components for the CAN Driver, the Network Management and your Application.

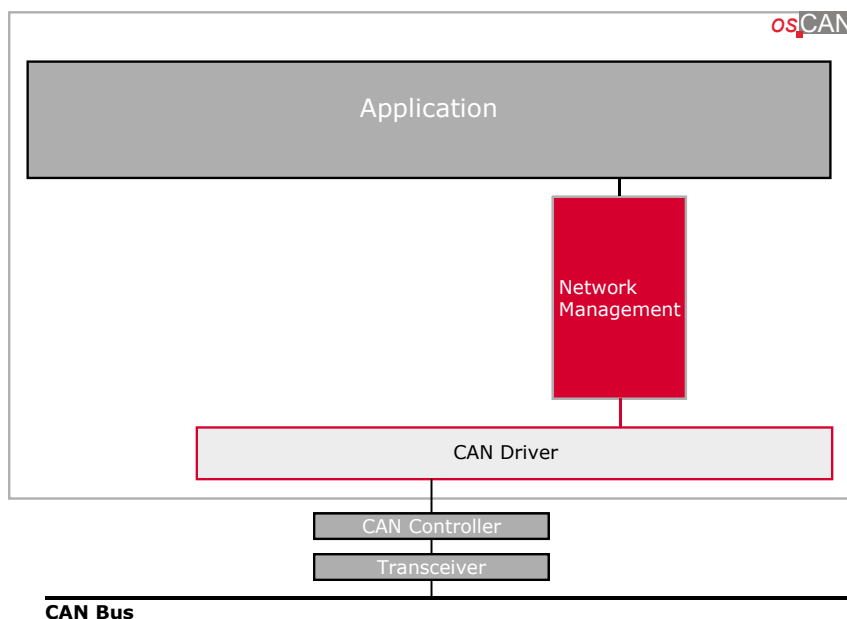


Figure 3-1 Overview CAN Driver, Network Management and Application

The OSEK Network Management is a software component to manage a CAN network and provides services for the user programs.

3.2 What the OSEK Network Management Does

The main tasks of the OSEK Network Management are:

- Controlling the transition of the nodes into the bus-sleep-mode.
- Recovery from Bus-Off state.
- Determine and update network configuration.
- Comparison of the network configuration.

The CANbedded modules from Vector follow the ISO OSI model. The higher layers use functionalities of the lower layers.

- Transfer of user data in the Network Management messages (ring data).
- Provide status information

3.3 Tools And Files

3.3.1 The Network Database

The dbc file is designed by the OEM (car manufacturer) and distributed to all suppliers that develop an ECU.

All settings necessary for the Network Management are usually done in the dbc file, delivered by your manufacturer.

3.3.2 Generation Tool

The Generation Tool is a PC-Tool. Using the information about your hardware/derivative and the dbc file and some further settings it generates files. These files contain information, macros and structures especially for your specific ECU and must be included into your application project.

Right now there are two Generation Tools available, CANgen and the new Generation Tool called GENy. Which tool you have to use depends on your delivery and the project.

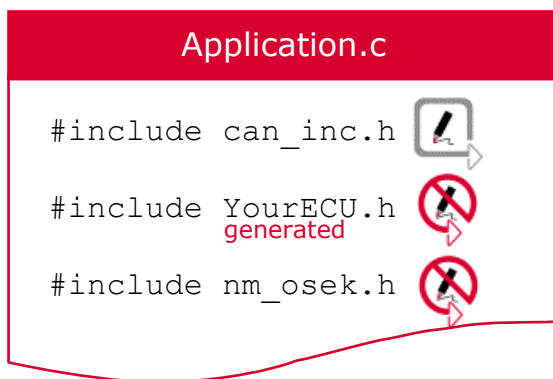
Include the generated files in your system as shown.

You must not change the generated files by hand – the next generation process will delete these changes.

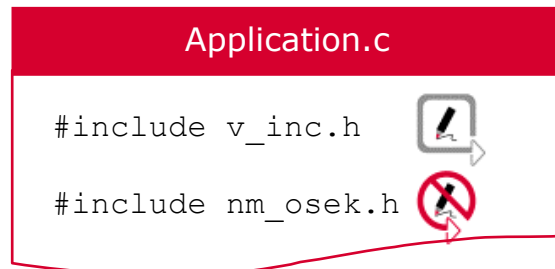
3.3.3 Include File Structure

Use the structure in the Figure 3-2 to include the files for the OSEK Network Management into your application correctly. Please keep to the including order to avoid errors while compiling and linking. The names can differ according to the manufacturer you are working for.

Using CANgen as Generation Tool



Using GENy as Generation Tool



Use the figure to get an overview of the modules which form the CAN Driver and the Network Management to do the includes correctly.

Figure 3-2 Include Files

3.4 Generation Process with CANbedded Software Component

Normally the Generation 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).

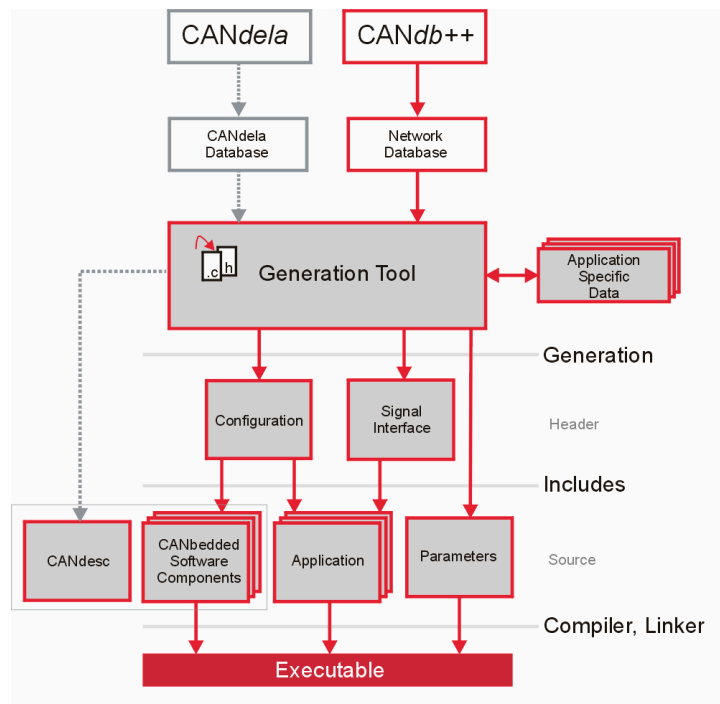


Figure 3-3 Generation Process For Vector CANbedded Software Components

The standard generation process for Vectors Software Components.

CANdesc is a completely generated Software Component.

4 OSEK Network Management – A More Detailed View

When you switch off the power in a company house, you must be sure that nobody is still working.

The same thing happens in a CAN Network within a modern vehicle. The ECUs exchange data to provide the other ECUs with the information they need. When you switch off e.g. the ignition you only can shut down the CAN communication when none of the ECUs participating in the network do need any further information from any other ECU.

From the network managements point of view the ECUs form a logical ring. Every ECU has its own network management message. The ID of this message determines the ring order. The network management follows the token ring principle.

Every ECU provides the information via its network management message, whether it needs the bus or not.

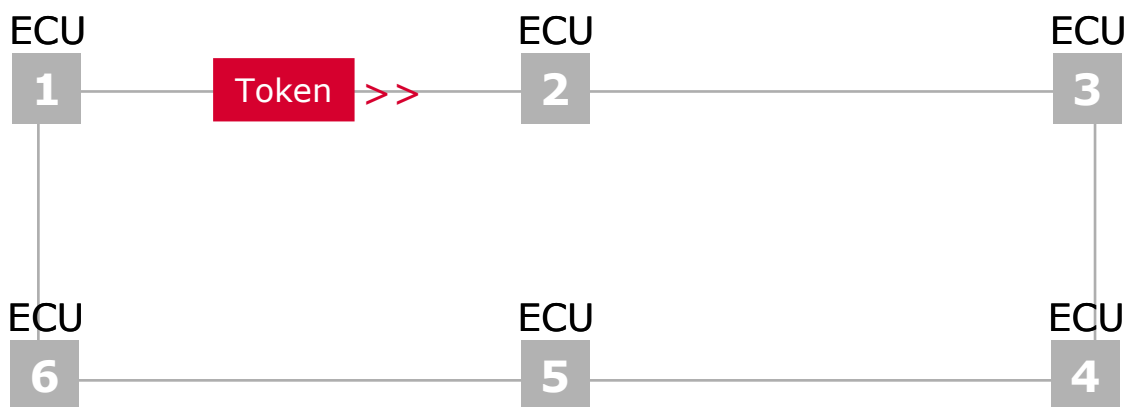


Figure 4-1 The ECUs form a ring via the token ring principle

The ECU with the token sends its network management message after a delay time and with it the token is sent to the next ECU in the logical ring. This minimizes the amount of Network Management messages on the bus, only one message all e.g. 100ms.

Remember the broadcast principle - a message is received by every ECU in the ring but the addressed one is the next one to send its Network Management message.

The ECUs participating in the Network Management form a logical ring and provide Network Management Information via their Network Management messages.

The token ring mechanism allows a controlled transition in the bus sleep state.



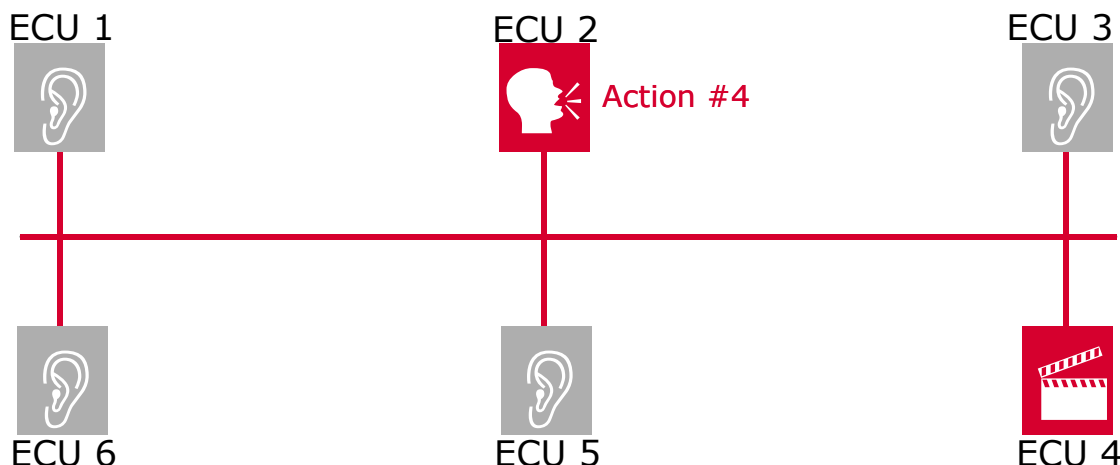


Figure 4-2 Broadcast Mechanism

The reason for this ring mechanism is the controlled transition into a bus sleep state. Just see the picture below with the contents of a Network Management message.



Figure 4-3 Example of contents of a Network Management Message. This is dependent on the vehicle manufacturer.

IDs of Network Management messages have a fix range, dependent on the car manufacturer.

The **DLC** is (as you know) the Data Length Code of the message and varies from 4 to 8 bytes, also dependent on the car manufacturer.

The **Destination** is the next ECU in the logical ring, the so-called successor.

The next byte contains information for the ring mechanism and the transition to sleep mode, the **OpCode**.

Ring: When the ring is established, every node has its successor the system is running.

Alive: Is for building up the ring in the startup phase.

LimpHome: When an ECU cannot receive or send messages, it goes in the limphome state.

BusSleepInd: An ECU that does not necessarily need the bus communication shows this to the other ECUs with the BusSleepInd.

With the NM message the ECU has to send the information, if it needs the bus or not (BusSleepInd). All other ECUs collect this information.

BusSleepAck: This flag is set if all ECUs are in the state BusSleepInd. After the token has done a complete rotation and every ECU has set the BusSleepInd-flag in its Network Management message, the BusSleepAck-flag is set. Via the broadcast mechanism all ECUs recognize the set BusSleepAck-flag and switch off the bus communication.

Only one ECU that does not set the BusSleepInd can keep the bus awake.

To reactivate the CAN Communication the ECU just has to activate its transceiver and the next CAN message will wake up all other ECUs and the ring will be set-up again.

4.1.1 Files to form the Network Management

The OSEK Network Management consists of 3 sorts of files.

The files for the CAN Driver and the general files are already listed in the User Manual for the CAN Driver and therefore not shown here again. The following files are only Network Management files.

4.1.1.1 Fix files that form the network management

Independent of the used Generation Tool

- nm_osek.c
- nm_osek.h

The names can differ dependent on the manufacturer.

4.1.1.2 Generated files that must not be changed, too

Independent of the used Generation Tool

- nm_cfg.h

Some information is also stored in the files YourECU.c and YourECU.h.

Only for GENy

- nm_par.h
- nm_par.c

The files are generated so do not change them because changes will be overwritten with the next generation process.



The Network Management consists of fix files, generated files and a file you should adapt.



The Network Management consists of fix files, generated files and a file you should adapt.

4.1.1.3 Configurable files

Independent of the used Generation Tool

- `can_inc.h`

4.1.1.4 `can_inc.h`

Independent of the used Generation Tool

INC stands for include. Here you can add includes you need (e.g. for usage of OSEK-OS). Now you have to include additionally the `nm_cfg.h`, you switched off before as you only needed the CAN Driver.

4.2 Handling of the OSEK Network Management

The OSEK Network Management has to be added to the application, initialized **after** the initialization of the CAN Driver. The `NmTask()` has to be called cyclically within the cycle time adjusted in the Generation Tool.

It is very important for the correct function of the OSEK Network Management that the timing you entered in the Generation Tool and the cycle you call the `NmTask()` are the same. This is the basis for the timing of the NM.

One of the values you can change is the `call cycle`. Take care that the value of this cycle is at least a fifth of a lowest time values and every timing value is a multiple of the `call cycle`.

E.g. there are the timing values from 100ms, 260ms, 1000ms and 1500ms. The lowest value is 100ms a fifth is 20ms. So, 20ms is the maximum value for the `call cycle`.



Additionally you need a timer for a cyclic call of the `NmTask()` to realize the timing tasks, the Network Management has to fulfil

5 OSEK Network Management In 7 Steps

STEP 1 : **UNPACK THE DELIVERY**

Figure out which component has to be placed at which memory location. Estimate the sizes.

STEP 2: **GENERATION TOOL AND DBC FILE**

This can be any application or the application you later use for the ECU. It is very important, that this application is running correctly and you can recognize its running. This application is the evidence for the correct function of the Flash Bootloader.

STEP 3: **GENERATE FILES**

In this step you have to integrate the Bootloader, download it via a programmer or burner and test it with the Flash Tool. This is the major step to be done!

Read the [Introduction of this step](#) or go directly to the [8 Bootloader Integration step](#).

STEP 4: **ADD FILES TO YOUR APPLICATION**

To prepare the application hex file for being downloaded via the Tester, it has to be converted to an OEM-specific format. Do this in this step and refer to your OEM-specific description.

STEP 5: **ADAPTATIONS FOR YOUR APPLICATION**

Now do the download the test application as before but now using the OEM-specific Tester.

STEP 6: **COMPILE AND LINK**

Now do the download the test application as before but now using the OEM-specific Tester

STEP 7: **TEST - CANOE AND SIMULATION**

Now do the download the test application as before but now using the OEM-specific Tester.



5.1 STEP 1 Unpack the Delivery

The delivery of CANbedded Software Components normally comprises a Generation Tool (CANgen or GENy) 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 to create a shortcut to the Generation Tool.

The Generation 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 7 Steps overview



5.2 STEP 2 Generation Tool and DBC File

Fist I recommend to copy the project for your first CAN Driver test into another directory and get it to work. Then do the changes for the NM when you have a base application, where CAN is working.

Normally the data base engineers do the adjustments in the data base (dbc file). To be able to use the OSEK Network Management the database attributes have to be set properly.

Use new Generation Tool GENy, look there >>

5.2.1 Using CANGen

Open the Generation Tool and add the dbc file as you did for the CAN Driver (see UserManual_CANDriver, remember the paths to generate in). Use the same settings, choose your target hardware and the node (YourECU) you want to generate for.

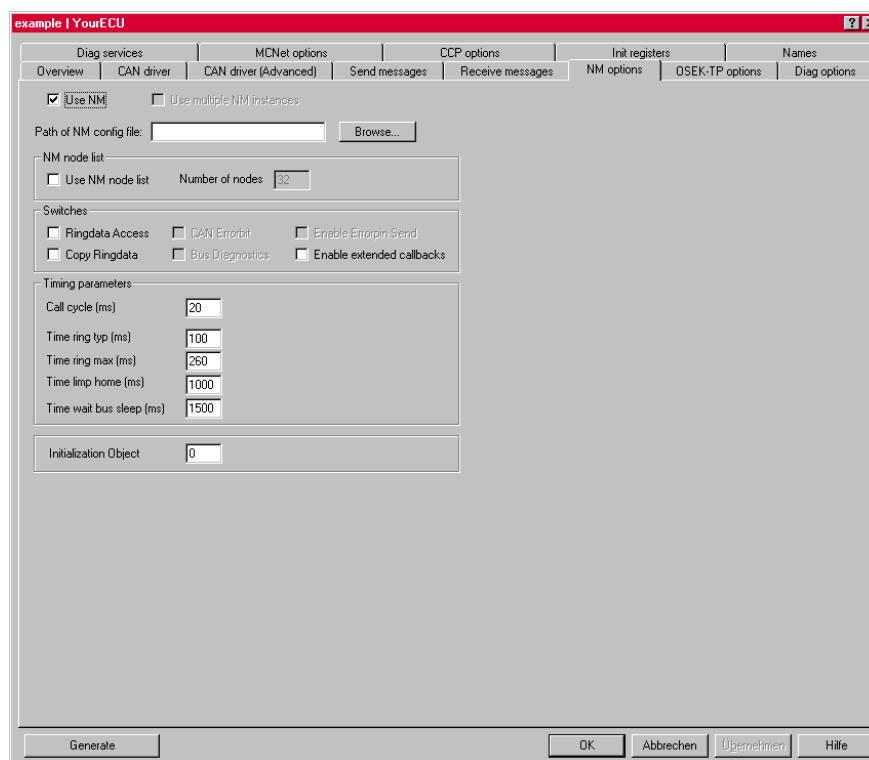


Figure 5-1 Settings for Network Management in the Generation Tool

For the first start of the Network Management do not change anything but switching off the **Use NM node list** button on this window.

Just make sure that the other modules like Transport Protocol, MCNet etc. are still switched off.

5.2.2 Using GENy

The following first steps with the Generation Tool are described in details in the online help of the Generation Tool, too.

Start the Generation Tool and setup a new configuration. Via **File/New** you open the Setup Dialog Window. Select License, Compiler, Micro and Derivative (if available) and confirm via **[OK]**.

Then open the Channel Setup Window via the green plus and the selection of the underlying bus system (CAN or LIN).

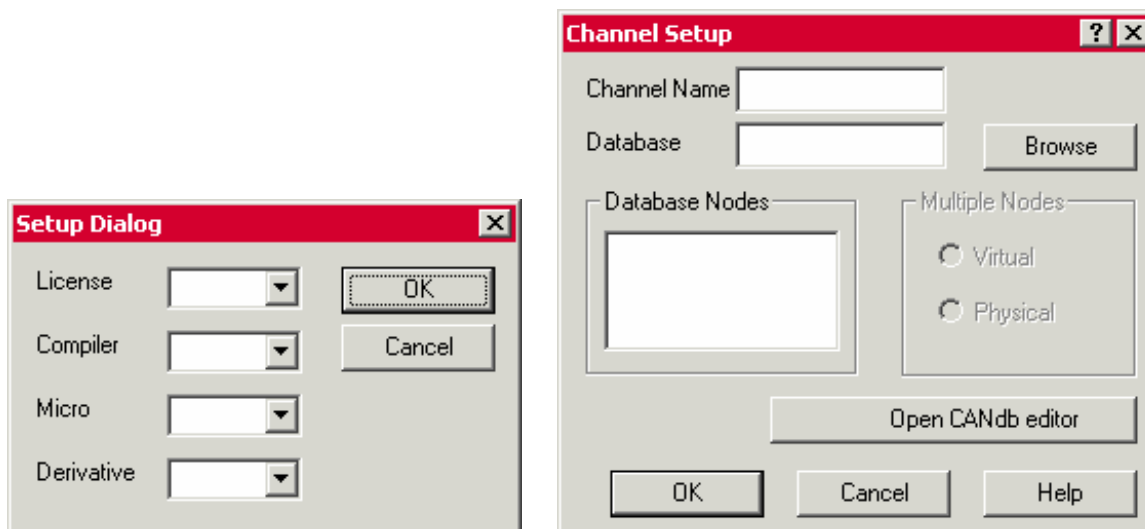


Figure 5-2 Setup Dialog Window and Channel Setup Window to Create a New Configuration

The channel name is Channel X per default (X ist starting with 0). Use the browse functionality to enter the location of the data base (dbc file).

Select you node out of the field Database Nodes and confirm the settings with **[OK]**.

Now save the configuration via **File/Save** or **File/Save as**.

First at all you should switch on/off the components you need, in this case we only use the CAN Driver and the OSEK Network Management.

Use the component selection (at the bottom of the main window of the Generation Tool) and select the suitable Driver (in this example application we use the CPUHC12 and the Driver HC12) and the OSEK Network Management (NM-Osek[direct]).

Now you should set the path where the Generation Tool generated the files to. To do this, open the Generation Directories Window via **Configuration/Generation Paths**. Enter the root path and select additionally individual paths for the components, if the Generation Tool should generated the files to different folders. This is also described very detailed in the GENy online help

You should be able to do the necessary settings for the CAN Driver. If not, refer to the UserManual_CANDriver.pdf.



For the first start of the Network Management you need not to set any checkbox.

Leave the channel-specific OSEK Network Management settings set to default values.


Remember the NM call cycle value, in this case 20ms.

[Back](#) to 7 Steps overview



5.3 STEP 3 Generate Files

5.3.1 Using CANgen Generation Tool

Click on the button  and start the generation process.

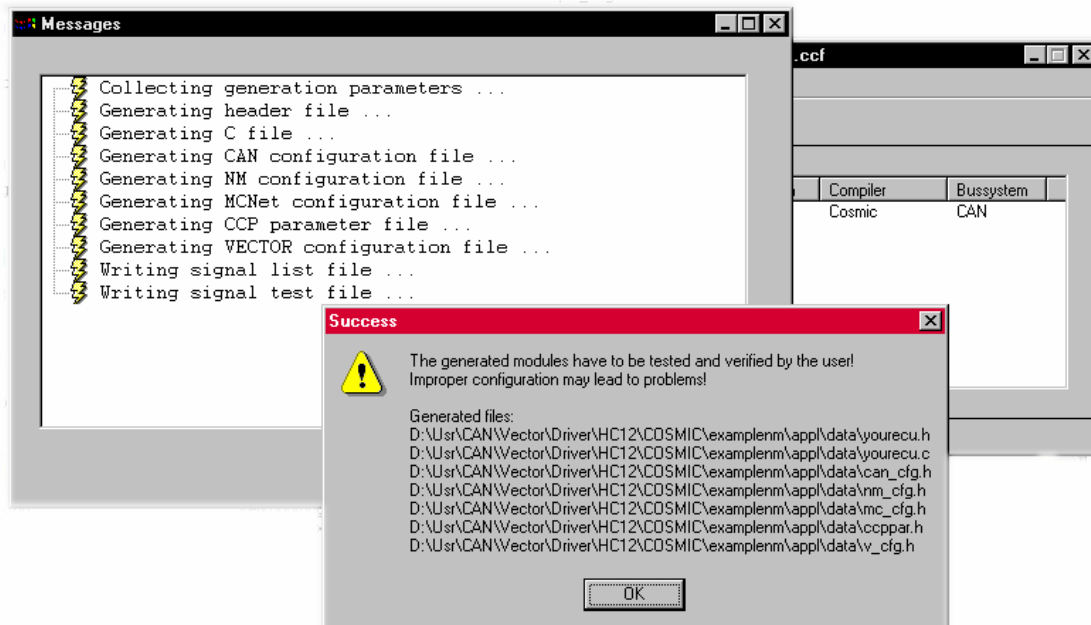


Figure 5-3 Generation Process

When you use the browse-button you will get an absolute path to the dbc-file.

I recommend that you use relative paths in order to be able to move your project more easily.

Remember to start the generation process after any change in the dialog windows of CANgen.

Check the directory and see the new files. Beneath the files as before there should be a file nm_cfg.h.

5.3.2 Using new Generation Tool

Click on the button  and start the generation process.

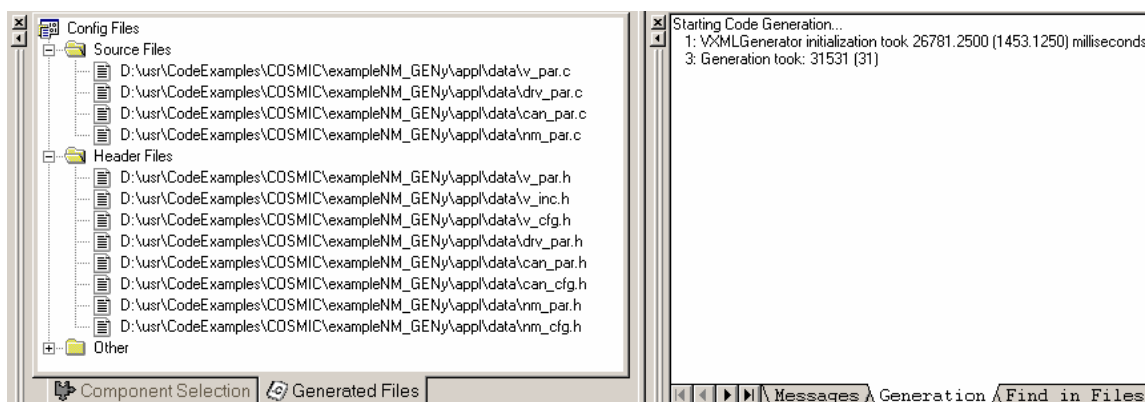


Figure 5-4 Information About the Generated Files and the Generation Process

The Generation Tool provides you with information about the **Generated Files** and **Generation** information.

Now we have generated for the first time. Check the directory and see the new files for the OSEK Network Management.

If you do not find the files check the paths in the **Generation Paths...** dialog.

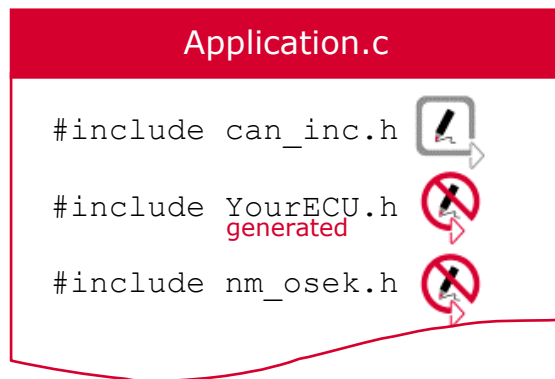
[Back](#) to 7 Steps overview



5.4 STEP 4 Add Files to your Application

Using the CANbedded Software Components (here the OSEK Network Management) is very simple as shown in the illustration below. There are only a few includes to be done in those C files you want to use component functionality.

Using CANGen as Generation Tool



Using GENy as Generation Tool

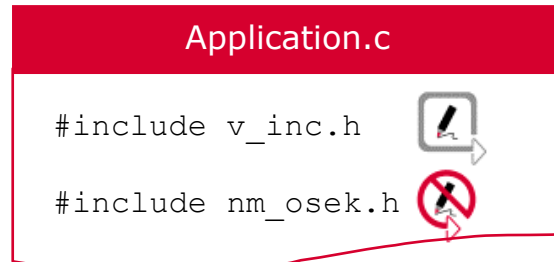


Figure 5-5 Include Components Into Your Application

Copy the Files for the Network Management (osek_nm.c and osek_nm.h, all other others should be generated to the correct location) to the directory you reserved for.

Now add the new files (component files and generated files) to your source list of your compiler or your makefile.

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

[Back](#) to 7 Steps overview



5.5 STEP 5 Adapt Your Application Files

To be able to compile and link, you have to do a few further adaptations in your application. This is the example you already know from the CAN Driver.

In the example a message is sent cyclically, controlled by the Interaction Layer. Another message should be received cyclically, if not a timeout occurs and a message is sent.

The includes in the following examples are designed for using CANGen as Generation Tool. To use the example for GENy, just adapt the include files as shown in the Figure 5-5.

Example for HC12:

```
/* Includes*****/
#include "can_inc.h"
#include "yourecu.h"
#include "nm_osek.h" /* Every c-file that has to use network
                      management functionality must include this
                      header*/

unsigned char timerelapsed; /*the function NmTask() 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, interrupts are disabled here, otherwise you have to
    disable the interrupts for initializations*/

    hardwareInit();
    CanInitPowerOn(kCanInitObj1);
    NmOsekInit(NM_NORMAL); /*This is to initialize the OSEK Network
                           Management. For the parameter refer to the
                           Technical Reference for the Network
                           Management. Keep the order of
                           initialization, first CAN Driver and then
                           the higher layers.*/

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

    enableInterrupts();
```



Refer to your hardware manual to get the information how to integrate a timer, to do the correct settings and to handle the timer interrupt.

Normally the interrupts are disabled short after an reset. But this depends on your hardware. Refer to your hardware manual.

```
for(;;)
{
    /* call of NmTask() if timerelapsed = 0xff*/
    if( timerelapsed == 0xff)
    {
        timerelapsed = 0;                /*clear the flag to call NmTask()
                                         only once after a timer
                                         interrupt*/

        NmTask();
    }
}

/* The function ApplCanBusOff() will be replaced by the following
function ApplNmBusOff because the Network Management controls the
handling of a bus off.*/

// void ApplCanBusOff( void )
// {
//     ; /*Call-back function for notification of BusOff*/
// }

/*The function is normally called from the CAN error ISR and indicates a
bus off state. But the bus off handling is again highly dependent on your
hardware.*/

void ApplNmBusOff( void )
{

}

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

/* The following functions are the call back functions of the Network
Management. We will have a closer look at the possible contents and the
context they are called in the Step 7 of this manual.*/

void ApplNmCanNormal( void )
{

}

void ApplNmCanSleep( void )
{

}

void ApplNmCanBusSleep( void )
{

}
```

```

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 variable timerelapsed indicates the occurrence of an
interrupt for the application (see code above).*/

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

```

The adjustments to realize a timer is up to you and very special depending on the hardware you use. Refer to your manual for this issue.

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[] = {
    CanTxInterrupt,      // @0xFFC4 start address of table
    CanRxInterrupt,      // $FFC4    CAN transmit
    CanErrorInterrupt,   // $FFC6    CAN receive
    irq_dummy0,          // $FFC8    CAN error
    irq_dummy0,          // reserved
    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
};
```

The next modification is to undo the delete of the `#include nm_cfg.h` as you did in Step 4 of the UserManual_CANDriver.

[Back](#) to 7 Steps overview

5.6 STEP 6 Compile And Link Your Project

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

[Back](#) to 7 Steps overview





5.7 STEP 7 Test It Via CANoe

Do you remember the testing method from the CAN Driver? To test the OSEK Network Management we use the same test environment with a few modifications.



Figure 5-6 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 (**CAN** open environment).

What you have to add is the Network Management for the TestNode. The Network Management for the simulated node is delivered as **DLL** and **ini-file**. Copy these files in the exec-path of your CANoe.

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

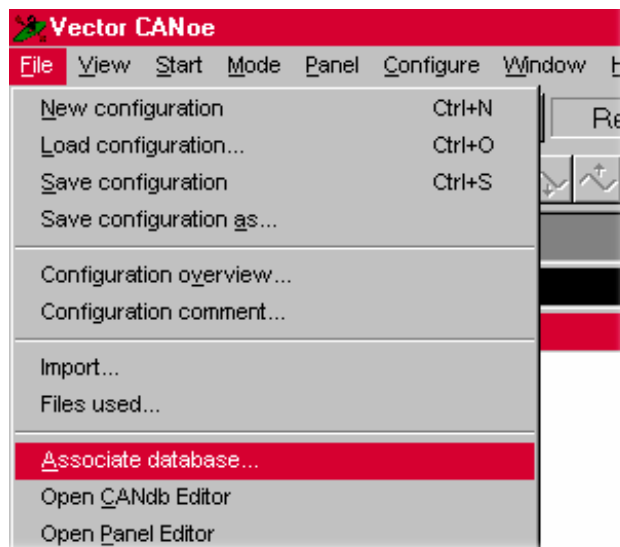


Figure 5-7 Add the database to CANoe

Insert a new network node using the right mouse button.

Use the same test environment as you did for the CAN Driver. You just have to add the Network Management DLL. The simulated node does not need any code for this test.

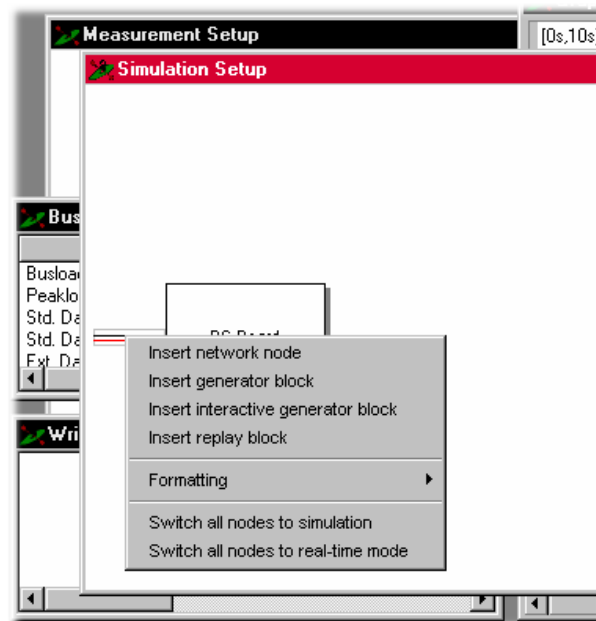


Figure 5-8 Insert a new network node

Then you have to open the configuration window of the new node by using the right mouse button, too. Select the name of the node and confirm with OK.

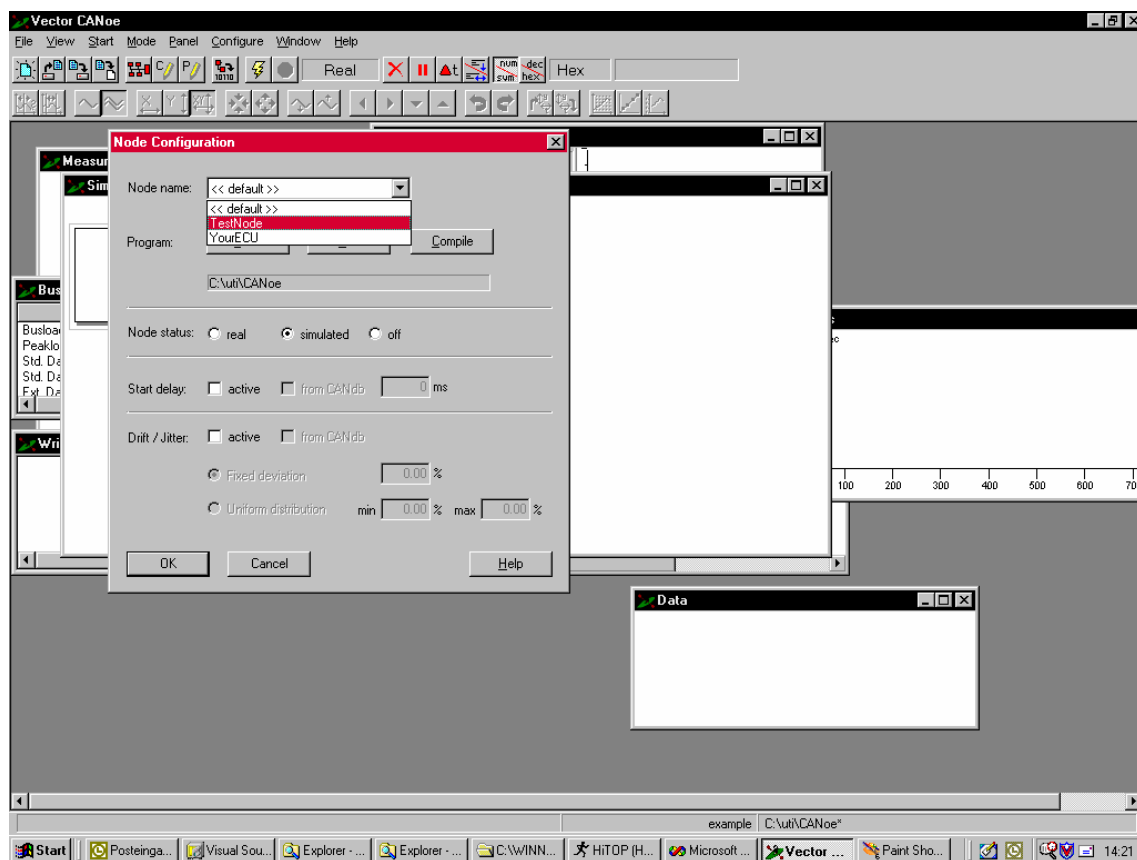


Figure 5-9 Node Configuration

When you confirm the latter adjustments you should see that the name of the new node has changed and that it is now a Network Management node (NM:xy).

Double-click this node and the CAPL-Browser will open up. Just click the save-symbol, chose an appropriate directory and close the browser again.

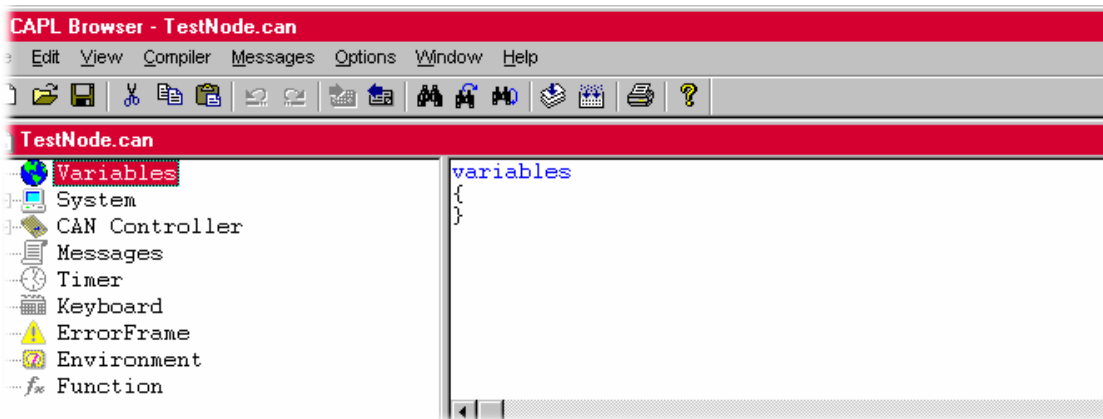


Figure 5-10 Save the node without any code

For the basic test of the Network Management we do not need any application in the simulated node.

Now compile all nodes (in the Configure Menu) and start the simulation.



Figure 5-11 Compile all nodes

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

Now start your Application on the hardware platform too. Do you see the messages from both sides?

!!! CONGRATULATIONS !!!

The Network Management is working, using the layer below, the CAN Driver. The 2 ECUs build up a ring, the time between the 2 of one message should be approximately 200ms, because of a typical ring time of 100ms for the Manufacturer Type **Vector**.

[Back](#) to 7 Steps overview

6 Further Information

Now the Network Management is basically working, but there is not any code in the provided call-back-functions. This step gives you an idea of the handling of the network via these functions.

You have to know about the different states of the Network Management and the transitions between them.

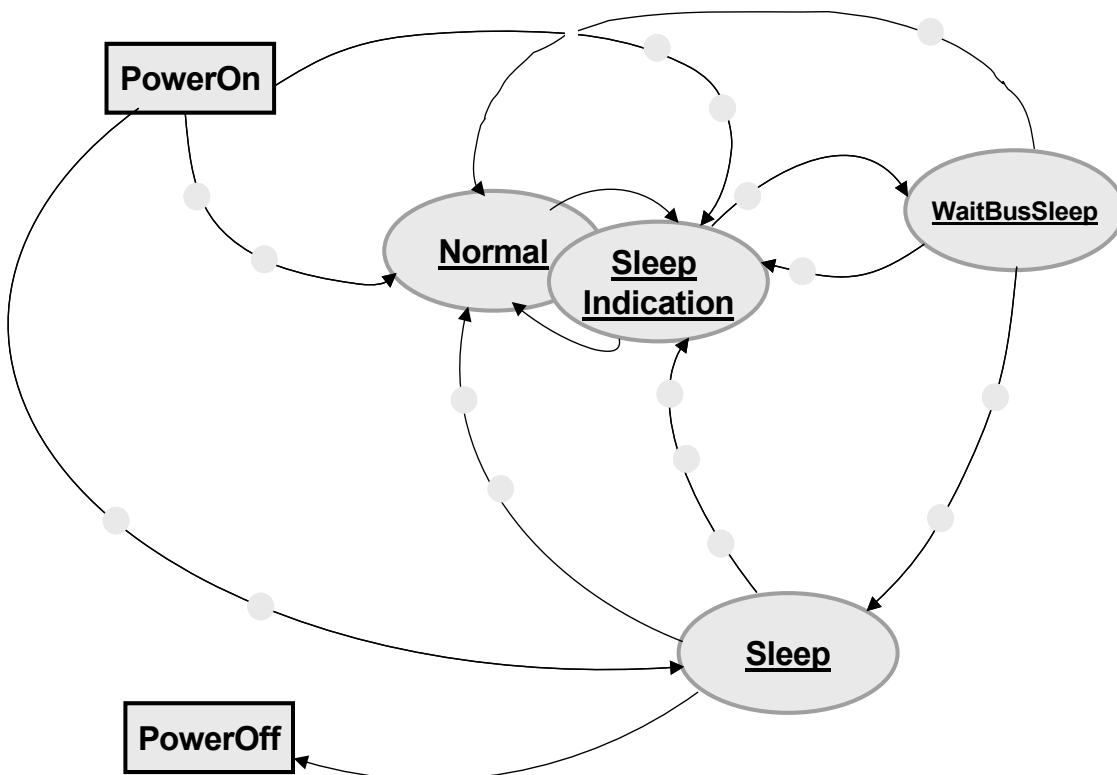


Figure 6-1 The states of the OSEK Network Management

You can read the information about the current state of the Network Management via the function `GetStatus()` or `NmGetStatus()`.

6.1.1 PowerOff / On

In the **PowerOff** the processor has no supply voltage and the CPU is off.

In the **PowerOn** the CPU is off too. The processor is in the power down or stop mode. Neither the application nor the network management are active.

Leaving this state is only possible by a HW-reset, an external wake up event or a CAN Bus activity.

6.1.2 Normal

The communication is active. The CAN Bus is not in Bus Sleep mode. The application is not ready to sleep.

6.1.3 Sleep Indication

The CAN-Bus is not in Bus sleep mode. The application has shown that it is ready to sleep by `NmOsekInit(NM_SLEEPIND)` or `GotoMode(BusSleep)`. The application is still allowed to send messages. If all ECUs are ready to sleep, the NM carries out the regular transition of the ECU into the local operation with BusSleep.

6.1.4 WaitBusSleep

The BusSleepAck is set and the bus is short before the sleep state. This state is to prevent the bus from waking up again via a CAN message that is in the controller just after the recognition of the BusSleepAck. In this state no application message will lead to a wake-up of the ECU.

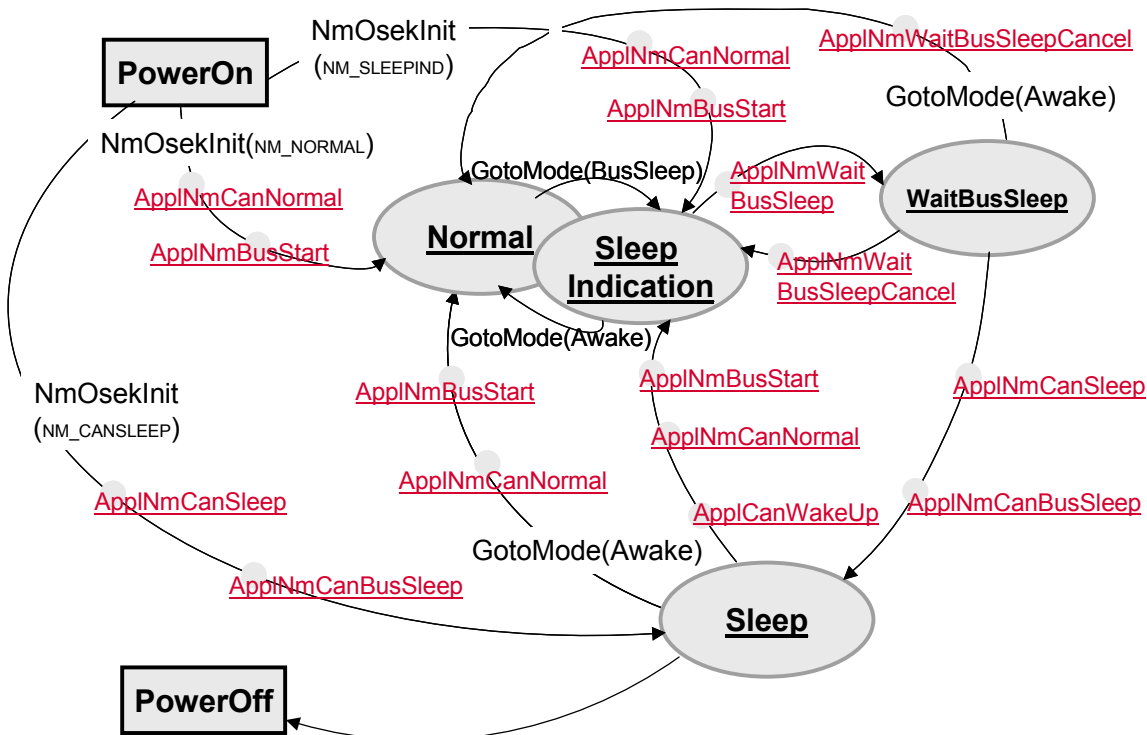
In WaitBusSleep state only NM messages can trigger a wake-up.

6.1.5 Sleep

The CAN-Bus is in Bus sleep mode. In this stage the CAN driver is locked for messages of the application and the NM (CanOffline).

Local applications can be active. The application has to check whether it requires the CAN Bus.

The next figure shows the transitions between the states in detail. It looks very complicated, but it gives you a good sight on what stands behind the OSEK Network Management and the order the functions are called.



Some functions listed here are called only if the extended callbacks are activated in the Generation Tool. See later, which function is extended.

Figure 6-2 The transitions between the state of the Network Management

Perhaps you do not recognize all service- and call-back-functions that form the transitions between the states, but most of them.

6.1.6 NmOsekInit(NM_SLEEPIND/NM_CANSLEEP/NM_NORMAL)

At the start-up of any software module, you have to do initial things, the so-called initialization. As you see there are 3 ways to start-up from PowerOn state, to **Normal**, **SleepIndication** and to **Sleep** state. The first 2 ways normally lead to a bus wakeup, the 3rd way allows just to wakeup the ECU without needing any bus communication.

This function has to be called by the application to initiate the transition.

6.1.7 GotoMode(Awake/BusSleep)

The GotoMode function triggers the transition to another state.

6.1.8 ApplNmCanNormal

This is the call-back-function for the application to react on the transition to Normal or SleepIndication state.

Use the little pictures to get an overview of the states of the different modules at the point in time the function is called

| Software | |
|---|-------------------|
| Network Management | normal / sleepind |
| CAN Driver | online |
| Hardware | |
| CAN Controller | normal |
| Bus Transceiver | Normal |
| Interrupt | |
| Called in NMOsekInit and NMTask, The INTERRUPT is locked during execution. | |

Table 6-1 States of the modules and hardware **after** execution of ApplNmCanNormal.

```
void ApplNmCanNormal(void)
{
    /* Switch Bus Transceiver to normal mode */
    BusTransceiverToNormal();

    /* Switch CAN Controller to normal mode */
}
```

```
CanWakeUp();
}
```

6.1.9 ApplNmCanSleep

This is the call-back-function for the application to react on the transition to Sleep state.

| Software | |
|---|---------|
| Network Management | sleep |
| CAN Driver | offline |
| Hardware | |
| CAN Controller | sleep |
| Bus Transceiver | Standby |
| Interrupt | |
| Called in NMOsekInit and NMTask, The INTERRUPT is locked during execution. | |

Table 6-2 States of the modules and hardware **after** execution of ApplNmCanSleep.

```
void ApplNmCanSleep(void)
{
    /* Switch CAN Controller to sleep mode */
    canuint8 returnCode;
    returnCode = CanSleep();

    if( returnCode == kCanFailed)
    {
        noSleep = 1; /*global variable*/
    }
    else
    {
        noSleep = 0; /*global variable*/
        /* Switch CAN Controller to standby mode */
        BusTransceiverToStandby( );
    }
}
```

6.1.10 ApplNmCanBusSleep

This is the call-back-function for the application to react on the transition to Sleep state.

| | |
|---|---------|
| Software | |
| Network Management | sleep |
| CAN Driver | offline |
| Hardware | |
| CAN Controller | sleep |
| Bus Transceiver | Sleep |
| Interrupt | |
| Called in NMOsekInit and NMTask, The INTERRUPT is locked during execution. | |

Table 6-3 States of the modules and hardware **after** execution of ApplNmCanBusSleep.

```
void ApplNmCanBusSleep(void)
{
    if( noSleep == 1 )
    {
        noSleep = 0; /*global variable*/
        GotoMode( Awake )
    }
    else
    {
        /* Switch CAN Controller to sleep mode */
        BusTransceiverToSleep( );
    }
}
```

This code is merely an example. The sleep handling is highly hardware dependent. Refer to your hardware manual to figure out the correct process for your application.



6.1.11 ApplCanWakeUp

The CAN Driver call this call-back-function if any activity on the CAN bus wakes up the controller.

| | |
|--|----------|
| Software | |
| Network Management | sleepind |
| CAN Driver | online |
| Hardware | |
| CAN Controller | normal |
| Bus Transceiver | Normal |
| Interrupt | |
| e.g. CAN Wakeup ISR | |
| Until now NMosekInit(NM_SLEEPIND) is called. | |

Table 6-4 States of the modules and hardware **after** execution of ApplCanWakeUp

```

void ApplCanWakeUp(void)
{
    /*Switch BUS Transceiver to normal mode*/
    BusTransceiverToNormal();

    /*Init OSEK NM*/
    NmOsekInit( NM_SLEEPIND );
}

```

6.1.12 Extended Callbacks

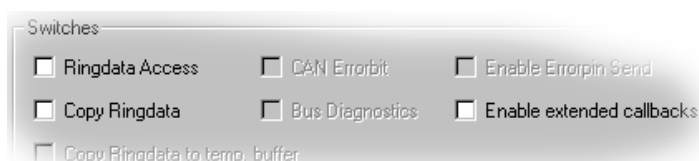


Figure 6-1 Extended Callbacks

On the tab NM options you can enable or disable the extended callbacks. These call-backs mainly are to serve higher layer modules (as e.g. the Interaction Layer) or for notification and are mentioned here to complete the list of call-back functions. Refer to the Technical Reference and the UserManual for the Interaction Layer to read more about these functions.

6.1.13 ApplNmWaitBusSleepCancel ()

This is the call-back-function for the application to react on the transition to Normal or SleepIndication state.

6.1.14 ApplNmWaitBusSleep ()

This is the call-back-function for the application to react on the transition to WaitBusSleep state.

6.1.15 ApplNmBusStart ()

This is the call-back-function for the application to react on the transition to Normal or SleepIndication state.

6.1.16 Handling of the Bus Transceivers

These functions are extremely dependent on your hardware, so just regard them as an example and **refer to your hardware description** for your special solution.

Refer to the Technical Reference for the Network Management for more details about the extended call back functions.

```
void BusTransceiverToNormal(void) {
    /*Switch transceiver to normal mode*/
    (PORT) STB = 1;
    (PORT) EN  = 1;
}
```

```
void BusTransceiverToStandby(void) {
    /*Switch transceiver to stand-by mode*/
    (PORT) EN  = 0;
    (PORT) STB = 0;
}
```

```
void BusTransceiverToSleep(void) {
    unsigned char waitCount;
    #define MAX_WAIT_LOOP 50    /*adapt this value to
                                processor speed */
    (PORT) EN = 1;
    /* make sure this loop is not removed by compiler optimization */
    for( waitCount = 0; waitCount < MAX_WAIT_LOOP; waitCount++ );    /*
    nothing to do - wait 50µs*/
    (PORT) EN = 0;
}
```

7 List Of Experiences

A list of experiences and problems encountered will be placed here soon, which will help you to conduct focused troubleshooting.

7.1 Topic 1

Q:

A: ■

8 Index

| | | | |
|---------------------------------------|------------|-------------------------------|-------------------------------|
| Alive | 14 | GotoMode | 36 |
| ApplCanWakeUp | 39 | Including Order | 11, 25 |
| ApplNmBusStart | 41 | ini-file | 30 |
| ApplNmCanBusSleep | 38 | initialization | 36 |
| ApplNmCanNormal | 36 | interface | 15 |
| ApplNmCanSleep | 37 | LimpHome | 14 |
| ApplNmWaitBusSleep | 41 | link | 26 |
| ApplNmWaitBusSleepCancel | 40 | Link | 29 |
| Awake | 36 | logical ring | 13 |
| Bootloader | 9 | makefile | 25, 29 |
| broadcast | 13 | Manufacturer Type | 32 |
| bus sleep state | 14 | MCNet | 19 |
| Bus Transceivers | 41 | Motivation | 3 |
| Bus-Off | 10 | network configuration | 11 |
| BusSleep | 36 | Network Management | 8 |
| BusSleepAck | 15, 35 | NM_CANSLEEP | 36 |
| BusSleepInd | 14, 15 | nm_cfg.h | 15, 16, 29 |
| bus-sleep-mode | 10 | NM_NORMAL | 36 |
| call-back-functions | 34, 36 | NM_SLEEPIND | 36 |
| can_def.h | 15 | NmGetStatus() | 34 |
| CANdesc IN 8 STEPS | 34 | NmOsekInit | 36 |
| CANdesc tab | 12 | Normal | 35 |
| clock | 29 | OSEK | 8 |
| compile | 26 | OSEK Network Management | 8, 10, 11, 15, 16, 19, 29, 35 |
| Compile | 29 | PowerOff | 35 |
| dbc-file | 11, 19, 30 | PowerOn | 34 |
| dbc-File | 10 | Ring | 14 |
| Destination | 14 | Sleep | 35 |
| Diagnostics | 10 | Sleep Indication | 35 |
| DLC | 14 | sleep mode | 14, 35, 38, 39 |
| DLL | 30 | states | 34, 36 |
| Example | 26, 28 | Test | 29 |
| gateway | 10 | token ring | 13 |
| generation process | 16, 23, 25 | transceiver | 15, 41 |
| Generation Process | 12 | transitions | 34, 35, 36 |
| GetStatus() | 34 | | |

| | | | |
|--------------------------|----|---------------------------|----|
| Transport Protocol | 19 | WaitBusSleep | 35 |
| user data | 11 | yourecu.h | 26 |