Nm_IndOsek

Technical Reference

Indirect Network Management

Version 1.12

Authors:	Markus Schwarz
Version:	1.12
Status:	released (in preparation/completed/inspected/released)

1 Document Information

1.1 History

Author	Date	Version	Remarks
Ralf Fritz	21.06.2001	1.00	Creation of this document
Ralf Fritz	01.10.2001	1.01	User value support added, Multiple ECU added
Dieter Schaufelberger	2002-02-28	1.02	ApplInmNmInitVolatileCounters( ) and Macros added
Dieter Schaufelberger	2002-08-15	1.03	Adoptions to new system structure
Dieter Schaufelberger	2002-09-11	1.04	Description of the database attributes revised
Dieter Schaufelberger	2002-10-22	1.05	Revision Inserted new chapter: Particularities of RENAULT Bus Off supervision
Dieter Schaufelberger	2002-11-28	1.06	Revision New configuration features
Dieter Schaufelberger	2003-07-31	1.07	Correction in the attribute part
Dieter Schaufelberger	2004-04-05	1.08	Inserted Support of LEVEL 3
Dieter Schaufelberger	2004-10-20	1.09	New OSEK_INM Version 2.0
Markus Schwarz	2006-09-12	1.10	Changed chapter 1.1, new layout
Markus Schwarz	2006-12-19	1.11	Revision and rework
Markus Schwarz	2008-01-21	1.12	added chapter on configuration with GENy

Table 1-1 History of the Document

1.2 Reference Documents

Index	Document
[UR_01]	OSEK/VDX Network Management 2.53
[UR_02]	Specification of the generic communication layers for CAN embedded networks at RENAULT & PSA Version 2.1 dated 04/01/98 referenced RENAULT: DIV/D3E/60601/98/032gb

Table 1-2 References Documents

1.3 Abbreviations & Acronyms

Abbreviation	Complete expression
CAN	Controller Area Network
ECU	Electronic Control Unit
IL	Interaction Layer
NM	Network Management Note: Within this document, NM refers to Nm_IndOsek
OEM	Original Equipment Manufacturer

Table 1-3 Abbreviations & acronyms

1.4 Naming Convention

Naming	Description
Nm_IndOsek	Refers to the Vector CANbedded software component that handles the indirect network management.

Table 1-4 Naming convention

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, Vector’s release of the programs delivered to your company is expressly restricted to the configuration you have specified in the questionnaire.

Contents

1 Document Information 2

1.1 History 2
1.2 Reference Documents 2
1.3 Abbreviations & Acronyms 3
1.4 Naming Convention 3

2 Overview 9

2.1 Delivery Package 9
2.2 Concept 10

3 Features 11

3.1 General 11
3.1.1 Overview 11
3.1.2 Control 11
3.1.3 Event Notification 11
3.1.4 Event Processing 11
3.1.5 Status Information 12
3.2 RX Supervision 13
3.2.1 Overview 13
3.2.2 Control 13
3.2.3 Event Notification 13
3.2.4 Status Information 13
3.3 TX Supervision 14
3.3.1 Overview 14
3.3.2 Control 14
3.3.3 Event Notification 14
3.3.4 Status Information 14
3.4 BusOff Supervision 15
3.4.1 Overview 15
3.4.2 Control 15
3.4.3 Event Notification 15
3.4.4 Status Information 15
3.4.5 Others 15
3.5 Generic Supervision 16
3.5.1 Overview 16
3.5.2 Control 16
3.5.3 Event Notification 16
3.5.4 Status Information 16

3.5.5 Others 16

4 Integration 17

4.1 Involved Files 17
4.2 Include Structure 17
4.3 Necessary Steps to Integrate the NM in Your Project 18
4.4 Necessary Steps to Run the NM 18

5 Configuration 19

6 Integration Hints 22

6.1 CANbedded stack 22
6.1.1 Vector Station Manager 22
6.2 Special use-cases 22
6.2.1 Multiple ECUs 22

7 Related Files 23

7.1 Static Files 23
7.2 Dynamic Files 23

8 API Description 24

8.1 General 24
8.1.1 Multi channel usage 24
8.2 API 25
8.2.1 NM Handler 26
8.2.2 RX Supervision 30
8.2.3 TX Supervision 33
8.2.4 BusOff Supervision 36
8.2.5 User-specific Supervision 39
8.3 Callbacks 42
8.3.1 NM Handler 43
8.3.2 RX Supervision 44
8.3.3 TX Supervision 46
8.3.4 BusOff Supervision 48
8.3.5 Generic Supervision 50
8.4 Other Interfaces 52
8.4.1 Version Information 52

9 Working with the Code 53

9.1 Version Information 53
9.2 Application Interface 53

10 CANdb Attributes 54

Illustrations

Figure 2-1 Concept of NM within the CANbedded stack 10
Figure 4-1 Include structure 17
Figure 5-1 System-specific configuration 19
Figure 5-2 Channel-specific configuration 21

Tables

Table 1-1 History of the Document 2
Table 1-2 References Documents 2
Table 1-3 Abbreviations & acronyms 3
Table 1-4 Naming convention 3
Table 3-1 Description of supervision error states 12
Table 5-1 System-specific configuration 20
Table 5-2 Channel-specific configuration 21
Table 8-1 Overview API 25
Table 8-2 Overview callbacks 42
Table 10-1 CANdb attributes 55

Introduction

This document describes the software component that implements the indirect network management.

The functionality of this NM is handled by the Vector CANbedded component “Nm_IndOsek”.

This document contains

- A brief description of the NM (chapter “3 Features”)
- A description of how to integrate the NM (chapter “4 Integration”)
- A description of how to configure the NM (chapter “5 Configuration”)
- A description of the API of the NM (chapter “8 API Description”)
- Hints on integration (chapter “6 Integration Hints“)

2 Overview

The component Nm_IndOsek provides network services for an application operating on a CAN bus. These services are mainly:

- Supervision of one TX message for each channel
- Supervision of multiple RX messages
- Supervision of BusOff events for each channel
- Supervision of user-specific events for each channel

The overall task of the Nm_IndOsek is to detect communication problems and to inform the application about these problems.

2.1 Delivery Package

The delivery package includes:

- source code and header files (see chapter “7 Related Files”)
- technical reference (this document)
- DLL for configuration tool GENy

2.2 Concept

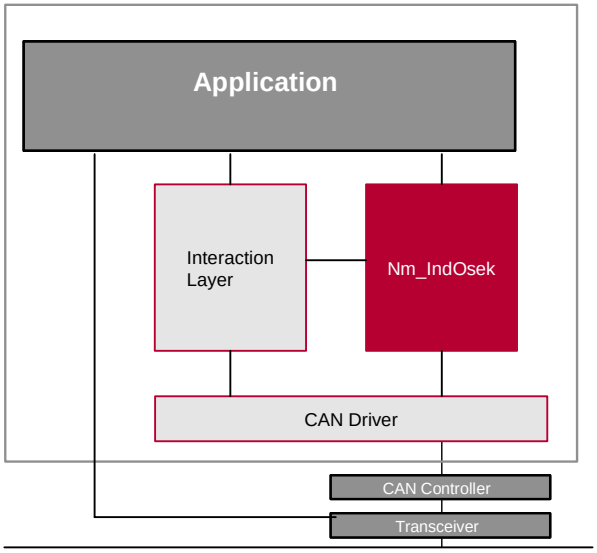

Figure 2-1 Concept of NM within the CANbedded stack

The Nm_IndOsek supervises events. These events can be signaled from the CAN driver, the IL and/or the application.

The NM is initialized and controlled by the application. Status changes are signalized with the help of callback functions. These callback functions have to be provided by the application (see chapter “8.3 Callbacks”).

The Nm_IndOsek can be used in following configurations

- Single channel ECUs
- Multi channel ECUs
- Multiple ECUs (refer to chapter “6.2.1 Multiple ECUs”)

3 Features

3.1 General

The Nm_IndOsek is an NM that uses periodic application messages to determine the network state.

It is a de-central NM. There is no NM master.

3.1.1 Overview

The Nm_IndOsek is an event manager that can handle the following events

- RX messages
- TX messages
- BusOff events
- User-specific events (=”Generic”)

The Nm_IndOsek does not monitor/detect the events itself. Instead, it only supervises and manages the state of these events.

It is up the IL and/or the application to inform the Nm_IndOsek about the occurrence/non-occurrence (=timeout) of an event.

If an event is absent for approx. 2 seconds, the event-related system part is confirmed absent.

If an event occurs again, the event-related system part is present again.

The Nm_IndOsek informs higher layers about the status of the supervised event. The higher layers can use this information to store diagnostic entries in non-volatile memory.

3.1.2 Control

The Nm_IndOsek can be started/sopped by the application (InmNmStart(), InmNmStop()).

Any supervision is only possible when the NM is running.

The Nm_IndOsek can start/stop all supervisions globally (InmNmDiagOn(), InmNmDiagOff()) as well as individually for each event handler.

3.1.3 Event Notification

The Nm_IndOsek offers an interface for each event handler that allows the components that detect an event (e.g. IL) to notify an event and/or an event timeout.

3.1.4 Event Processing

When an event or an event timeout is notified, the Nm_IndOsek updates the status for this event. The status consists of a supervision counter and a related error state.

The supervision counter is incremented upon an event timeout and decremented upon an event notification. The values for the increment/decrement are mostly generated by the

configuration tool. In case of RX or TX message events, these values are derived from the message attributes in the DBC file (see chapter “10 CANdb Attributes”).

The values are chosen in a way that any event is reported confirmed absent approx. 2 seconds after its last occurrence.

The error state is related to the supervision counter (see Table 3-1).

State	Description
OK	This state is entered when an event is notified. Note: This state can occur even if the supervision counter is not 0.
Failure	This state is entered when an event timeout is notified.
Confirmed Failure	This state is entered when there is no notification of the event for approx. 2 seconds, i.e. the supervision counter reaches its upper limit (0xFF).

Table 3-1 Description of supervision error states

3.1.5 Status Information

The Nm_IndOsek offers an interface that can be used to retrieve the current status of each event handler individually.

The application is also notified about any status change with the help of a callback that is unique for each event type.

3.2 RX Supervision

The Nm_IndOsek implements a supervision for RX messages.

3.2.1 Overview

The RX supervision is a message-specific event supervision. The RX supervision is always enabled.

There can be multiple supervised RX messages. These messages are marked in the DBC file (see chapter “10 CANdb Attributes”).

There should be only one supervised RX message for each network node that is meant to be supervised.

All APIs related to RX supervision have a parameter “index”. This parameter is a handle of the relevant RX message that has to be supervised. The index itself is provided by the ECU configuration file (e.g. board1.h).

3.2.2 Control

The RX supervision gets activated/de-activated together with the NM generic supervision (InmNmDiagOn(),InmNmDiagOff()).

There is an API that activates/de-activates the RX supervision individually, independent of the NM generic supervision state (InmNmRxDiagOn(), InmNmRxDiagOff()).

3.2.3 Event Notification

The Nm_IndOsek offers an interface that allows the system to notify a RX event (InmNmRxOk()) and a RX timeout event (InmNmRxTimeOut()).

3.2.4 Status Information

The application can retrieve the current status of the TX supervision (InmNmRxTimeOut()).The status information contains the error state and the value of the supervision counter.

The application is also notified about any status change with the help of callback ApplInmNmStatusIndicationRx().

3.3 TX Supervision

The Nm_IndOsek implements a supervision for TX messages.

3.3.1 Overview

The TX supervision is a channel-specific event supervision. The TX supervision is always enabled.

There can be only one supervised TX message for each channel. This message is marked in the DBC file (see chapter “10 CANdb Attributes”).

3.3.2 Control

The TX supervision gets activated/de-activated together with the NM generic supervision state (InmNmDiagOn(),InmNmDiagOff()).

There is an API that activates/de-activates the TX supervision individually, independent of the NM generic supervision (InmNmTxDiagOn(), InmNmTxDiagOff()).

3.3.3 Event Notification

The Nm_IndOsek offers an interface that allows the system to notify a TX event (InmNmTxOk()) and a TX timeout event (InmNmTxTimeOut()).

3.3.4 Status Information

The application can retrieve the current status of the TX supervision (InmNmGetTxCondition()).The status information contains the error state and the value of the supervision counter.

The application is also notified about any status change with the help of callback ApplInmNmStatusIndicationTx().

3.4 BusOff Supervision

The Nm_IndOsek implements a BusOff supervision and a notification mechanism for BusOff recovery.

The Nm_IndOsek does not do any BusOff recovery itself!

3.4.1 Overview

The BusOff supervision is a channel-specific event supervision. This supervision can be enabled/disabled within the component configuration, i.e. during configuration time.

Any data or API is only available and relevant if this feature is enabled.

3.4.2 Control

The BusOff supervision gets activated/de-activated together with the NM generic supervision (InmNmDiagOn(),InmNmDiagOff()).

There is an API that can activate/de-activate the BusOff supervision individually, independent of the NM generic supervision state (InmNmBusOffDiagOn(), InmNmBusOffDiagOff()).

3.4.3 Event Notification

The Nm_IndOsek offers an interface that allows the system to notify a BusOff event (InmNmBusOff()). This event is typically signaled by the CAN driver and can occur on interrupt level.

There is no API to indicate a non-BusOff. The disappearance of the BusOff is detected within the component Nm_IndOsek as soon as successful RX or TX event is reported.

3.4.4 Status Information

The application can retrieve the current status of the BusOff supervision (InmNmGetBusOffStatus()).

The application is also notified about any status change with the help of callback ApplInmNmStatusIndicationBusOff().

3.4.5 Others

As the BusOff supervision is based on a timer and not directly on an event counter, the Nm_IndOsek needs a constant time base. This time base is provided by the cyclic task function InmNmTask(). This function has to be called cyclically by the application with the constant period defined within the configuration (see chapter “5 Configuration”).

3.5 Generic Supervision

The Nm_IndOsek implements a supervision for a user-specific event. This user-specific event is called “GenericUser”

3.5.1 Overview

The GenericUser supervision is a channel-specific event supervision. This supervision can be enabled/disabled within the component configuration, i.e. during configuration time.

Any data or API is only available and relevant if this feature is enabled.

3.5.2 Control

The GenericUser supervision gets activated/de-activated together with the NM generic supervision (InmNmDiagOn(),InmNmDiagOff()).

There is an API that can activate/de-activate the GenericUser supervision individually, independent of the NM generic supervision (InmNmGenericDiagOn(), InmNmGenericDiagOff()).

3.5.3 Event Notification

The Nm_IndOsek offers an interface that allows the system to notify a GenericUser event (InmNmGenericOk(), InmNmGenericTimeOut()).

3.5.4 Status Information

The application can retrieve the current status of the GenericUser supervision (InmNmGetGenericCondition()). The status information contains the error state and the value of the supervision counter.

The application is also notified about any status change with the help of callback ApplInmNmStatusIndicationGeneric().

3.5.5 Others

The GenericUser supervision has to be configured by the application.

The application has to provide the following arrays:

```
V_MEMROM0 extern V_MEMROM1 inmNmCounterType V_MEMROM2 inmkIncGeneric[INM_CHANNELS];
```

```
V_MEMROM0 extern V_MEMROM1 inmNmCounterType V_MEMROM2 inmkDecGeneric[INM_CHANNELS];
```

These arrays must contain the channel-specific values that are used to increment/decrement the supervision counter in case the application reports an event or a timeout of the event.

4 Integration

4.1 Involved Files

To integrate the NM in your project, you need the files that are listed in chapter “7 Related Files”.

4.2 Include Structure

The include structure of the involved files is shown in Figure 4-1.

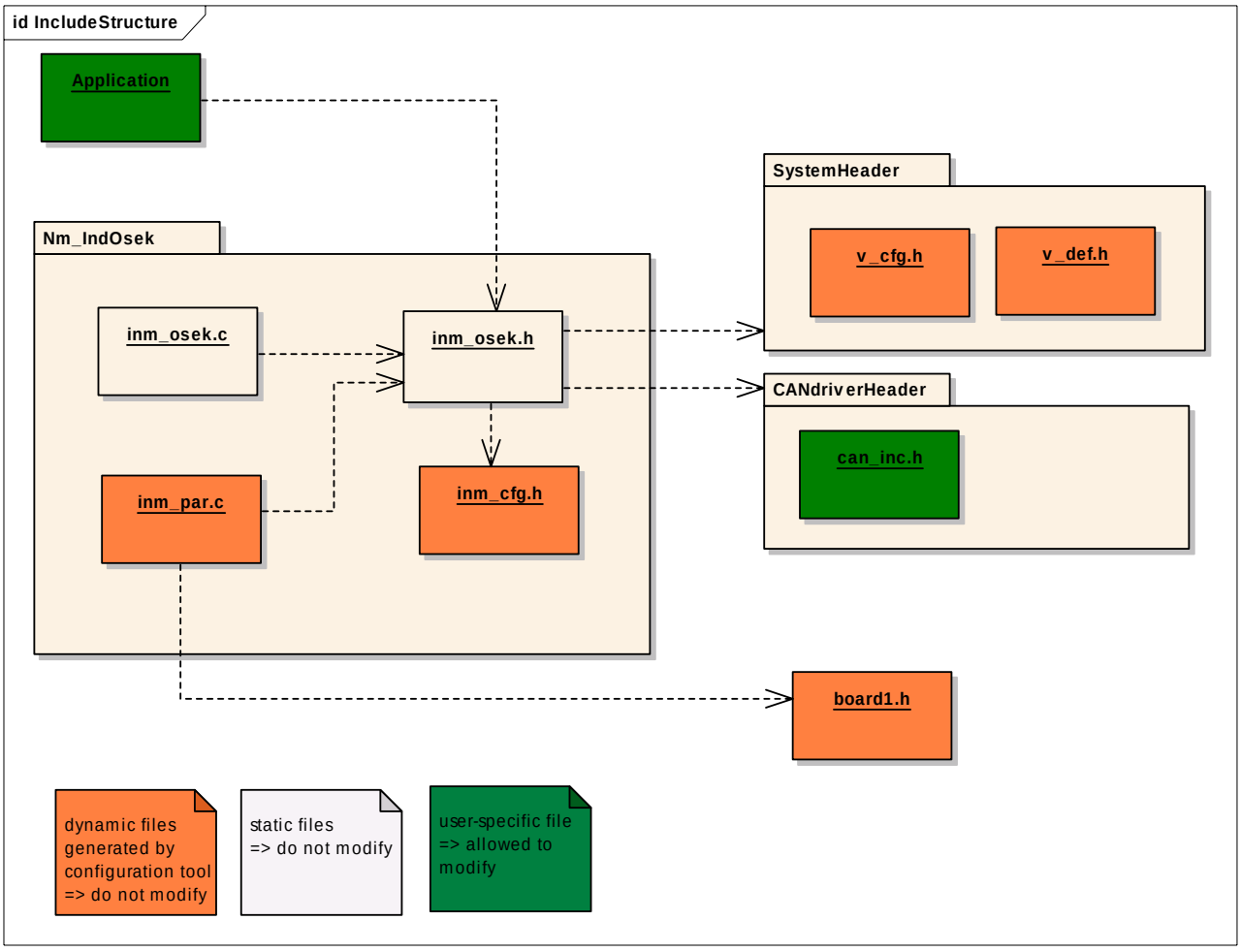

Figure 4-1 Include structure

4.3 Necessary Steps to Integrate the NM in Your Project

Following steps may be necessary to integrate the NM in your project:

- Copy the NM related files into your project tree.
- Make these files available in your project settings, e.g. set the correct paths in your makefile.
- In order to make the NM available to your application, include the component header file (inm_osek.h) into all files that make use of NM services and functions.
- Configure the NM according to your needs (see chapter “5 Configuration”).
- Implement all necessary callbacks in your application (see chapter “8.3 Callbacks”).
- Build your project (compile & link).

4.4 Necessary Steps to Run the NM

The NM should already have been integrated in your project and the building process should complete without any errors.

There are two main steps that have to be performed:

Initialization

- Initialize the CAN Driver by calling CanInitPowerOn() after each reset during start-up and before initializing the NM. Interrupts have to be disabled until the complete initialization procedure is done.
- Initialize the NM by calling InmNmInit() during start-up. This API has to be called for each NM channel separately.

Cyclic call of task functions

- If you are using the BusOff supervision, add a cyclic function call of InmNmTask() to your runtime environment. Ensure that the call cycle matches the value that is configured in the configuration file.

Add the required NM services to your application.

Especially use the API InmNmStart()and InmNmStop() to start and stop the Nm_IndOsek.

5 Configuration

The configuration data is stored in the following files:

- Inm_cfg.h contains system-specific value definitions and defines that enable/disable system-specific features
- Inm_par.c contains channel-specific value definitions

These files are created with the help of a PC-based configuration tool.

Settings for the NM can be selected in the GUI. These settings are used to generate the configuration files, which are needed to compile the component.

The configuration options for Nm_IndOsek are mainly derived from the settings in the DBC file. There are not much options in the GUI.

Note: The Nm_IndOsek is configured together with an OEM-specific station manager.

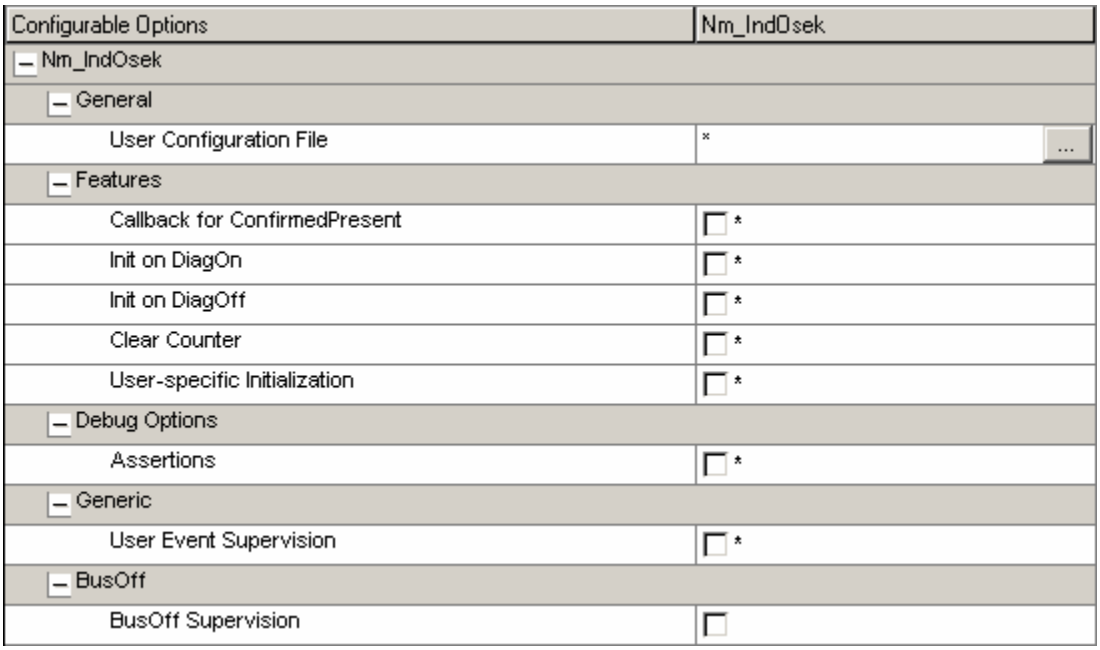

Figure 5-1 System-specific configuration

There are some configurations options that are directly related to the attribute definitions in the DBC file, mostly the used OEM. These configuration options can’t be changed in the GUI of the configuration tool.

Attribute	Description
User Configuration File	A configuration file is generated by GENy. If you want to overwrite settings in the generated configuration file, you can specify a path to a user defined configuration file. The user defined configuration file will be included at the end of the generated file. Therefore definitions in the user defined configuration file can overwrite definitions in the generated configuration file.
Callback for ConfirmedPresent	Enables/disables a feature that reports the presence of a supervised event not before it gets" confirmed present", i.e. the supervision counter reaches 0. If enabled, the callback for status indication is only executed when the corresponding error counter reaches 0. If disabled, the callback is executed each time when the corresponding error counter is decremented.
Init on DiagOn	Enables/disables a feature that re-initializes the status of a supervised event when the supervision for this event is started.
Init on DiagOff	Enables/disables a feature that re-initializes the status of a supervised event when the supervision for this event is stopped.
Clear Counter	Enables/disables a feature that confirms the presence of an event upon the first detection of that event, i.e. the event supervision counter is immediately set to 0 when the event happens. If enabled, the error counter of the sub module (RX, TX, Generic) is reset when the message/event is reported present the first time. If disabled, the error counter is decremented upon each reported presence.
User-specific initialization	Enables/disables a feature that lets the application set the (re-)initialization status of the supervised events. Instead of using the internal default values, the application is notified to set the initial status within callbacks AppIInm…UserInit() and AppIInmNm…UserReInit() If enabled, the initialization and re-initialization of the sub modules (RX, TX, Generic, BusOff) can be done by the application. The application is notified by callbacks about the need for the corresponding (re-)initialization. If disabled, the Nm_IndOsek (re-)initializes the status itself to the (internal) default values.
Assertions	The SW component provides built-in debug support (assertion) to ease up the integration and test into the user's project. Please see the technical reference for detailed information on the available options and how to use them. In general, the usage of assertions is recommended during the integration and pre-test phases. It is not recommended to enable the assertions in production code due to increased runtime and ROM needs. The assertion checks the correctness of the assigned condition and calls an error handler in case this fails. The error handler is called with an error number. Information about the defined error numbers is given in the technical reference.
User Event Supervision	Enables/disables a feature to supervise user-specific events. The increment/decrement values for event occurrence are channel-specific, i.e. they are configured within the channel-specific settings.
BusOff Supervision	Enables/disables a feature to supervise BusOff events.

Table 5-1 System-specific configuration

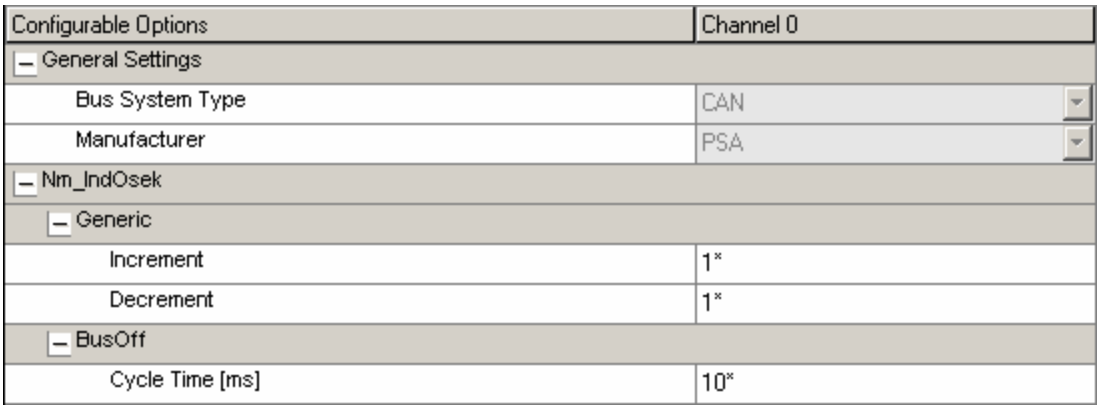

Figure 5-2 Channel-specific configuration

Attribute	Description
Increment	Defines the increment value that is used when the user-specific event is detected as absent (timeout of event). This value can be calculated with the folllowing formula: (<Time_EventTimeout> / 2000) * 255 This value is only configurable if the system-specific NM attribute 'User Event Supervision' is enabled.
Decrement	Defines the increment value that is used when the user-specific event is detected as present (occurrence of event). This value can be calculated with the following formula: (<Time_EventCycle> / 2000) * 255 This value is only configurable if the system-specific NM attribute 'User Event Supervision' is enabled.
Cycle Time [ms]	Defines the cycle time of the NM task function in [ms]. This value is only configurable when the system-specific NM attribute 'BusOff Supervision' is enabled, as task is only required for BusOff supervision. Please ensure that the configured value matches the actual call cycle of the task function in your system.

Table 5-2 Channel-specific configuration

6 Integration Hints

6.1 CANbedded stack

6.1.1 Vector Station Manager

The Vector Station Manager uses the Nm_IndOsek.

The Station Manager completely covers the necessary interfaces and control methods for the Nm_IndOsek. The Station Manager

- Handles the RX timeout monitoring
- Handles the TX timeout monitoring
- Handles the initialization and the re-initialization
- Handles the start/stop of the Nm_IndOsek

6.2 Special use-cases

6.2.1 Multiple ECUs

The Nm_IndOsek supports multiple ECUs. A multiple ECU contains more than one ECU. The currently active ECU can be determined dynamically at power-on.

The Nm_IndOsek of multiple ECUs handles the RX and TX messages of all ECU instances in a common algorithm.

The Nm_IndOsek uses only one TX supervision. As there are multiple supervised TX messages when using multiple ECUs, the Nm_IndOsek would react on all event notifications of any supervised TX message. Therefore the IL has to ensure that the timeout notifications of all TX messages that are not relevant for the current ECU are not processed.

Please refer to the documentation of the IL for more details.

However, it is not necessary to stop the event notification of irrelevant RX messages within the IL. These RX messages might still be processed by the Nm_IndOsek. It is up to the application to ignore the related status indications of those RX messages.

7 Related Files

7.1 Static Files

Inm_osek.c	This is the source file of the NM. It contains all API and algorithms.
Inm_osek.h	This is the header file of the NM. It contains all static prototypes for the API and definitions, such as symbolic constants.

Caution
It is not allowed to change the NM source code during the integration into the application. Please contact the Vector hotline if any problems arise.

7.2 Dynamic Files

The dynamic files are created by the configuration tool (see chapter “5 Configuration”).

inm_cfg.h	This is the configuration file for system-specific items of the NM. It defines features and certain values of the NM.
inm_par.c	This is the configuration file for channel-specific items of the NM. It defines features and certain values of the NM.

Caution
Do not change anything within the dynamic files, as the changes will be overwritten when the configuration tool generates these files the next time.

8 API Description

The API of the NM consists of services that are realized by function calls. These services can be called whenever they are required. They transfer information to the NM or take over information from the NM.

If not stated otherwise, the API of the NM is not re-entrant.

Info
A function is re-entrant if it can be safely called recursively or from multiple processes.

Note
The application functions must match the required interfaces.

This can be ensured by including the header file *.h in the modules which provide the required application functions. If these interfaces do not match, unexpected run-time behavior may occur.

These functions are not allowed to change the interrupt status.

8.1 General

8.1.1 Multi channel usage

Most API functions of multi-channel systems require a channel parameter.

The channel parameter inm_channel represents an NM-internal channel. Possible values are: 0…max(NM channel).

Note
The Nm_IndOsek uses internal NM channels within the parameter list.

8.2 API

	NM	RX supervision	TX supervision	BusOff supervision	User-specific supervision
Init	■ InmNmInit() ■ InmNmReInit()				
Control	■ InmNmStart() ■ InmNmStop()				
Control supervision	■ InmNmDiagOn() ■ InmNmDiagOff()	■ InmNmRxDiagOn() ■ InmNmRxDiagOff()	■ InmNmTxDiagOn() ■ InmNmTxDiagOff()	■ InmNmBusOffDiagOn() ■ InmNmBusOffDiagOff()	■ InmNmGenericDiagOn() ■ InmNmGenericDiagOff()
Status information	■ InmNmGetStatus()	■ InmNmGetRxCondition()	■ InmNmGetTxCondition()	■ InmNmGetBusOffStatus()	■ InmNmGetGenericCondition()
Event notification		■ InmNmRxOk() ■ InmNmRxTimeOut()	■ InmNmTxOk() ■ InmNmTxTimeOut()	■ InmNmBusOff()	■ InmNmGenericOk() ■ InmNmGenericTimeOut()
Others				■ InmNmTask()	

Table 8-1 Overview API

8.2.1 NM Handler

InmNmInit()

Prototype	
Single channel	`void InmNmInit ( void )`
Multi channel	`void InmNmInit ( inmNmChannelType inm_channel )`
Parameter	
inm_channel	Handle of the NM-internal channel. Range: 0…max(NM channels)
Return code	
---	---
Functional Description	
This function initializes the component Nm_IndOsek. All internal states and variables are initialised. This function has to be called once after power-on reset, while interrupts are disabled. This function initializes all available sub-components.	
Particularities and Limitations	
■ Has to be called with disabled CAN interrupts.	

InmNmReInit()

Prototype	
Single channel	`void InmNmReInit ( void )`
Multi channel	`void InmNmReInit ( inmNmChannelType inm_channel )`
Parameter	
inm_channel	Handle of the NM-internal channel. Range: 0…max(NM channels)
Return code	
---	---
Functional Description	
This function re-initializes the Nm_IndOsek. Only some internal states and variables are re-initialised. A re-initialization does not affect the status of the NM handler, i.e. the NM itself stays in the same state as before (started or stopped). This function re-initializes all available sub-components.	
Particularities and Limitations	
■ The NM has to be initialized before.	

InmNmStart()

Prototype	
Single channel	`void InmNmStart ( void )`
Multi channel	`void InmNmStart ( inmNmChannelType inm_channel )`
Parameter	
inm_channel	Handle of the NM-internal channel. Range: 0…max(NM channels)
Return code	
---	---
Functional Description	
This function starts the Nm_IndOsek if it is not already running. The application is informed about the current status of all available sub-components with the help of the corresponding callback functions. The application is informed about the start of the NM with the help of callback ApplInmNmStartCanIl().	
Particularities and Limitations	
■ The NM has to be initialized before.	

InmNmStop()

Prototype	
Single channel	`void InmNmStop ( void )`
Multi channel	`void InmNmStop ( inmNmChannelType inm_channel )`
Parameter	
inm_channel	Handle of the NM-internal channel. Range: 0…max(NM channels)
Return code	
---	---
Functional Description	
This function stops the Nm_IndOsek if it is not already stopped. The supervision of all available sub-components is stopped. The application is informed about the stop of the NM with the help of callback ApplInmNmStopCanIl().	
Particularities and Limitations	
■ The NM has to be initialized before.	

InmNmDiagOn()

Prototype	
Single channel	`void InmNmDiagOn ( void )`
Multi channel	`void InmNmDiagOn ( inmNmChannelType inm_channel )`
Parameter	
inm_channel	Handle of the NM-internal channel. Range: 0…max(NM channels)
Return code	
---	---
Functional Description	
This function starts the network supervision for all available sub-components. Starting the supervision is only possible if the NM is started.	
Particularities and Limitations	
■ The NM has to be started before (=>InmNmStart() has to be called before)	

InmNmDiagOff()

Prototype	
Single channel	`void InmNmDiagOff ( void )`
Multi channel	`void InmNmDiagOff ( inmNmChannelType inm_channel )`
Parameter	
inm_channel	Handle of the NM-internal channel. Range: 0…max(NM channels)
Return code	
---	---
Functional Description	
This function stops the network supervision for all available sub-components. Stopping the supervision is only possible if the NM is started. If the NM gets stopped (InmNmStop()), the supervision is also stopped automatically.	
Particularities and Limitations	
■ The NM has to be started before (=>InmNmStart() has to be called before)	

InmNmGetStatus()

Prototype	
Single channel	`inmNmStatusType` **`InmNmGetStatus`** `( void )`
Multi channel	`inmNmStatusType` **`InmNmGetStatus`** `( inmNmChannelType inm_channel )`
Parameter	
inm_channel	Handle of the NM-internal channel. Range: 0…max(NM channels)
Return code	
`inmNmStatusType`	Status of NM: INM_NM_ON : NM is running INM_NM_OFF: NM is not running
Functional Description	
This function returns the current status of the NM.	
Particularities and Limitations	
■ The NM has to be initialized before.	

8.2.2 RX Supervision

InmNmRxDiagOn()

Prototype	
Single channel	`void InmNmRxDiagOn ( const inmNmIndexType index )`
Multi channel	`void InmNmRxDiagOn ( const inmNmIndexType index )`
Parameter	
index	Handle of the supervised RX message.
Return code	

Functional Description	
This function starts the supervision of the RX message that is given by parameter <index>. This message is under supervision until the supervision is stopped (InmNmRxDiagOff() or InmNmDiagOff()) or the NM is stopped (InmNmStop()). This function has only effect if the NM is running, i.e. InmNmStart() has been called. It is not possible to enable the supervision when the NM is stopped.	
Particularities and Limitations	
■ Dependent on the OEM, the supervision status gets re-initialized when supervision is started.	

InmNmRxDiagOff()

Prototype	
Single channel	`void InmNmRxDiagOff ( const inmNmIndexType index )`
Multi channel	`void InmNmRxDiagOff ( const inmNmIndexType index )`
Parameter	
index	Handle of the supervised RX message.
Return code	

Functional Description	
This function stops the supervision of the RX message that is given by parameter <index>. The supervision is automatically stopped when all supervisions are stopped (InmNmDiagOff())) or when the NM is stopped (InmNmStop()).	
Particularities and Limitations	
■ Dependent on the OEM, the supervision status gets re-initialized when supervision is stopped.	

InmNmRxOk()

Prototype	
Single channel	`void InmNmRxOk ( const inmNmIndexType index )`
Multi channel	`void InmNmRxOk ( const inmNmIndexType index )`
Parameter	
index	Handle of the supervised RX message.
Return code	

Functional Description	
Successful RX notification This function is normally called by the interaction layer. It informs the INM OSEK about an successfully reception of an observed message. In this function the volatile and non volatile reception and Bus-Off counter are reduced depending on the current state.	
Particularities and Limitations	
■	

InmNmRxTimeOut()

Prototype	
Single channel	`void InmNmRxTimeOut ( const inmNmIndexType index )`
Multi channel	`void InmNmRxTimeOut ( const inmNmIndexType index )`
Parameter	
index	Handle of the supervised RX message.
Return code	

Functional Description	
This function is normally called by the interaction layer. It informs the INM OSEK that an observed message can't be received during a certain time. In this function the volatile and non volatile reception counter are reduced depending on the current state.	
Particularities and Limitations	
■	

InmNmGetRxCondition()

Prototype	
Single channel	`inmNmConditionType *InmNmRxTimeOut ( const inmNmIndexType index )`
Multi channel	`inmNmConditionType *InmNmRxTimeOut ( const inmNmIndexType index )`
Parameter	
index	Handle of the supervised RX message.
Return code	
`inmNmConditionType`	Reference to condition struct *inmNmConditionType*
Functional Description	
This function returns the current condition of the supervised ECU. The values can be modified by the application.	
Particularities and Limitations	
■	

8.2.3 TX Supervision

InmNmTxDiagOn()

Prototype	
Single channel	`void InmNmTxDiagOn ( void )`
Multi channel	`void InmNmTxDiagOn ( inmNmChannelType inm_channel )`
Parameter	
inm_channel	Handle of the NM-internal channel. Range: 0…max(NM channels)
Return code	

Functional Description	
This function starts the supervision of the TX message. This message is under supervision until the supervision is stopped (InmNmTxDiagOff()or InmNmDiagOff()) or the NM is stopped (InmNmStop()). This function has only effect if the NM is running, i.e. InmNmStart() has been called. It is not possible to enable the supervision when the NM is stopped.	
Particularities and Limitations	
■ Dependent on the OEM, the supervision status gets re-initialized when supervision is started.	

InmNmTxDiagOff()

Prototype	
Single channel	`void InmNmTxDiagOff ( void )`
Multi channel	`void InmNmTxDiagOff ( inmNmChannelType inm_channel )`
Parameter	
inm_channel	Handle of the NM-internal channel. Range: 0…max(NM channels)
Return code	

Functional Description	
This function stops the supervision of the TX message. The supervision is automatically stopped when all supervisions are stopped (InmNmDiagOff())) or when the NM is stopped (InmNmStop()).	
Particularities and Limitations	
■ Dependent on the OEM, the supervision status gets re-initialized when supervision is stopped.	

InmNmTxOk()

Prototype	
Single channel	`void InmNmTxOk ( void )`
Multi channel	`void InmNmTxOk ( inmNmChannelType inm_channel )`
Parameter	
inm_channel	Handle of the NM-internal channel. Range: 0…max(NM channels)
Return code	
Functional Description	
Transmit successful notification This function will normally called by the interaction layer. It informs the INM OSEK about an successfully transmission of the message which should be observed.	
Particularities and Limitations	
■	

InmNmTxTimeOut()

Prototype	
Single channel	`void InmNmTxTimeout ( void )`
Multi channel	`void InmNmTxTimeout ( inmNmChannelType inm_channel )`
Parameter	
inm_channel	Handle of the NM-internal channel. Range: 0…max(NM channels)
Return code	
`---`	
Functional Description	
Transmit time out notification This function is normally called by the interaction layer. It informs the INM OSEK that the observed message couldn't be send at the moment. The volatile counter is incremented to his maximum and state NM_FAILURE is set. If maximum is reached state NM_CONFIRMED_FAILURE is entered and the non volatile counter is set to his maximum. This function must be called cyclically if the message could not be sent. The cycle time of calling this function is the cycle time of the observed message.	
Particularities and Limitations	
■	

InmNmGetTxCondition()

Prototype	
Single channel	`inmNmConditionType *InmNmGetTxCondition ( void )`
Multi channel	`inmNmConditionType *InmNmGetTxCondition ( inmNmChannelType inm_channel )`
Parameter	
inm_channel	Handle of the NM-internal channel. Range: 0…max(NM channels)
Return code	
`inmNmConditionType`	Reference to condition struct *inmNmConditionType*
Functional Description	
Transmit time out notification This function is normally called by the interaction layer. It informs the INM OSEK that the observed message couldn't be send at the moment. The volatile counter is incremented to his maximum and state NM_FAILURE is set. If maximum is reached state NM_CONFIRMED_FAILURE is entered and the non volatile counter is set to his maximum. This function must be called cyclically if the message could not be sent. The cycle time of calling this function is the cycle time of the observed message.	
Particularities and Limitations	
■	

8.2.4 BusOff Supervision

InmNmBusOffDiagOn()

Prototype	
Single channel	`void InmNmBusOffDiagOn ( void )`
Multi channel	`void InmNmBusOffDiagOn ( inmNmChannelType inm_channel )`
Parameter	
inm_channel	Handle of the NM-internal channel. Range: 0…max(NM channels)
Return code	

Functional Description	
This function starts the channel-specific supervision of the BusOff event. This supervision runs until the supervision is stopped (InmNmBusOffDiagOff() or InmNmDiagOff()) or the NM is stopped (InmNmStop()). This function has only effect if the NM is running, i.e. InmNmStart() has been called. It is not possible to enable the supervision when the NM is stopped.	
Particularities and Limitations	
■ Dependent on the OEM, the supervision status gets re-initialized when supervision is started.	

InmNmBusOffDiagOff()

Prototype	
Single channel	`void InmNmBusOffDiagOff ( void )`
Multi channel	`void InmNmBusOffDiagOff ( inmNmChannelType inm_channel )`
Parameter	
inm_channel	Handle of the NM-internal channel. Range: 0…max(NM channels)
Return code	

Functional Description	
This function stops the channel-specific supervision of the BusOff event. The supervision is automatically stopped when all supervisions are stopped (InmNmDiagOff())) or when the NM is stopped (InmNmStop()).	
Particularities and Limitations	
■ Dependent on the OEM, the supervision status gets re-initialized when supervision is stopped.	

InmNmBusOff()

Prototype	
Single channel	`void InmNmBusOff ( void )`
Multi channel	`void InmNmBusOff ( inmNmChannelType inm_channel )`
Parameter	
inm_channel	Handle of the NM-internal channel. Range: 0…max(NM channels)
Return code	
`---`	
Functional Description	
Bus Off notification form CAN driver This function is called by the CAN-Driver if an Bus-Off happened. The application is informed about this by calling the function ApplInmNmBusOffIndication(...). The volatile Bus-Off counter will be incremented until the maximal value is reached. If the maximal value is reached the state NM_CONFIRMED_FAILURE will be entered and the non volatile counter will be set to his maximum.	
Particularities and Limitations	
■	

InmNmTask()

Prototype	
Single channel	`void InmNmTask ( void )`
Multi channel	`void InmNmTask ( inmNmChannelType inm_channel )`
Parameter	
inm_channel	Handle of the NM-internal channel. Range: 0…max(NM channels)
Return code	
`---`	
Functional Description	
Cyclic Task for timeout supervision of Bus Off	
Particularities and Limitations	
■ Call context: ■ Availability:	

InmNmGetBusOffStatus()

Prototype	
Single channel	`inmNmStatusType *InmNmGetBusOffStatus ( void )`
Multi channel	`inmNmStatusType * InmNmGetBusOffStatus ( inmNmChannelType inm_channel )`
Parameter	
inm_channel	Handle of the NM-internal channel. Range: 0…max(NM channels)
Return code	
`inmNmStatusType`	Reference to condition struct *inmNmStatusType*
Functional Description	
This function returns the current state of the Network status. The values can be modified by the application. For bus off Supervision no counter is used. Therefore only the states *OK, Failure and Confirmed Failure* are handled.	
Particularities and Limitations	
■	

8.2.5 User-specific Supervision

InmNmGenericDiagOn()

Prototype	
Single channel	`void InmNmGenericDiagOn ( void )`
Multi channel	`void InmNmGenericDiagOn ( inmNmChannelType inm_channel )`
Parameter	
inm_channel	Handle of the NM-internal channel. Range: 0…max(NM channels)
Return code	

Functional Description	
This function starts the supervision of the user-specific event. This event is under supervision until the supervision is stopped (InmNmGenericDiagOff() or InmNmDiagOff()) or the NM is stopped (InmNmStop()). This function has only effect if the NM is running, i.e. InmNmStart() has been called. It is not possible to enable the supervision when the NM is stopped.	
Particularities and Limitations	
■ Dependent on the OEM, the supervision status gets re-initialized when supervision is started.	

InmNmGenericDiagOff()

Prototype	
Single channel	`void InmNmGenericDiagOff ( void )`
Multi channel	`void InmNmGenericDiagOff ( inmNmChannelType inm_channel )`
Parameter	
inm_channel	Handle of the NM-internal channel. Range: 0…max(NM channels)
Return code	

Functional Description	
This function stops the supervision of the user-specific event. The supervision is automatically stopped when all supervisions are stopped (InmNmDiagOff())) or when the NM is stopped (InmNmStop()).	
Particularities and Limitations	
■ Dependent on the OEM, the supervision status gets re-initialized when supervision is stopped.	

InmNmGenericOk()

Prototype	
Single channel	`void InmNmGenericOk ( void )`
Multi channel	`void InmNmGenericOk ( inmNmChannelType inm_channel )`
Parameter	
inm_channel	Handle of the NM-internal channel. Range: 0…max(NM channels)
Return code	
Functional Description	
Transmit successful notification This function will normally called by the interaction layer. It informs the INM OSEK about an successfully transmission of the message which should be observed.	
Particularities and Limitations	
■	

InmNmGenericTimeOut()

Prototype	
Single channel	`void InmNmGenericTimeout ( void )`
Multi channel	`void InmNmGenericTimeout ( inmNmChannelType inm_channel )`
Parameter	
inm_channel	Handle of the NM-internal channel. Range: 0…max(NM channels)
Return code	
`---`	
Functional Description	
Transmit time out notification This function is normally called by the interaction layer. It informs the INM OSEK that the observed message couldn't be send at the moment. The volatile counter is incremented to his maximum and state NM_FAILURE is set. If maximum is reached state NM_CONFIRMED_FAILURE is entered and the non volatile counter is set to his maximum. This function must be called cyclically if the message could not be sent. The cycle time of calling this function is the cycle time of the observed message.	
Particularities and Limitations	
■	

InmNmGetGenericCondition()

Prototype	
Single channel	`inmNmConditionType *InmNmGetGenericCondition ( void )`
Multi channel	`inmNmConditionType *InmNmGetGenericCondition ( inmNmChannelType inm_channel )`
Parameter	
inm_channel	Handle of the NM-internal channel. Range: 0…max(NM channels)
Return code	
`inmNmConditionType`	Reference to condition struct *inmNmConditionType*
Functional Description	

Transmit time out notification

This function is normally called by the interaction layer. It informs the INM OSEK that the observed message couldn't be send at the moment.

The volatile counter is incremented to his maximum and state NM_FAILURE is set. If maximum is reached state NM_CONFIRMED_FAILURE is entered and the non volatile counter is set to his maximum.

This function must be called cyclically if the message could not be sent. The cycle time of calling this function is the cycle time of the observed message.

Particularities and Limitations

-

8.3 Callbacks

The Nm_IndOsek uses callback functions to inform the application about an event. It is up to the application to react on this event.

The application has to implement the necessary callbacks. The prototypes of the callbacks are listed in the NM *.h-file.

There are following types of callbacks (also see Table 8-2):

- Callbacks that are used to inform the application that a certain event supervision has to be (re-)initialized. These callbacks require actions by the application: The application has to set the initial values, i.e. the value of the supervision counter and the supervision state. These callbacks are only available if the user-specific configuration is enabled in the configuration.
- Callbacks that are used to inform the application about status changes. These callbacks are only meant for notification/information. The application might use these callbacks to e.g. store the current status in non-volatile memory. The NM does depend on the actions done by the application. These callbacks are always available.
- Callbacks that are meant to be used to control other CANbedded modules. These callbacks are always available.

	Initialization	Status information	Others
NM handler			■ ApplInmNmStartCanIl() ■ ApplInmNmStopCanIl()
RX super-vision	■ ApplInmNmRxUserInit() ■ ApplInmNmRxUserReInit()	■ ApplInmNmStatusIndicationRx()	
TX super-vision	■ ApplInmNmTxUserInit() ■ ApplInmNmTxUserReInit()	■ ApplInmNmStatusIndicationTx()	
BusOff super-vision	■ ApplInmNmBusOffUserInit() ■ ApplInmNmBusOffUserReInit()	■ ApplInmNmStatusIndicationBusOff()	
Generic super-vision	■ ApplInmNmGenericUserInit() ■ ApplInmNmGenericUserReInit()	■ ApplInmNmStatusIndicationGeneric()	

Table 8-2 Overview callbacks

8.3.1 NM Handler

ApplInmNmStartCanIl()

Prototype	
Single channel	`void ApplInmNmStartCanIl ( void )`
Multi channel	`void ApplInmNmStartCanIl ( inmNmChannelType inm_channel )`
Parameter	
inm_channel	Handle of the NM-internal channel. Range: 0…max(NM channels)
Return code	

Functional Description	
This callback informs the application that the Nm_IndOsek has been started. This callback can be used to start the CAN driver and the IL.	
Particularities and Limitations	
■ Call context: task level	

ApplInmNmStopCanIl()

Prototype	
Single channel	`void ApplInmNmStopCanIl ( void )`
Multi channel	`void ApplInmNmStopCanIl ( inmNmChannelType inm_channel )`
Parameter	
inm_channel	Handle of the NM-internal channel. Range: 0…max(NM channels)
Return code	

Functional Description	
This callback informs the application that the Nm_IndOsek has been stopped. This callback can be used to stop the CAN driver and the IL.	
Particularities and Limitations	
■ Call context: task level	

8.3.2 RX Supervision

ApplInmNmStatusIndicationRx()

Prototype	
Single channel	`void` **`ApplInmNmStatusIndicationRx`** `( inmNmIndexType index, inmNmStatusType status )`
Multi channel	`void` **`ApplInmNmStatusIndicationRx`** `( inmNmIndexType index, inmNmStatusType status )`
Parameter	
index	handle of related RX message
status	status of the RX supervision ■ INM_OK ■ INM_FAILURE ■ INM_CONFIRMED_FAILURE ■ INM_SPV_ACTIVE
Return code	

Functional Description	
This callback is executed when the status of the supervision for the RX message with handle <index> changes, i.e. when the call of InmNmRxOk() or InmNmRxTimeOut() leads to a change of the supervision state. This callback is also executed during system initialization and re-initialization. The new status is contained in the parameter <status>.	
Particularities and Limitations	
■ Call context: maybe in IR context, depending on call context of InmNmRxOk() and InmNmRxTimeOut().	

ApplInmNmRxUserInit()

Prototype	
Single channel	`void` **`ApplInmNmRxUserInit`** `( inmNmIndexType index, inmNmConditionType *condition )`
Multi channel	`void` **`ApplInmNmRxUserInit`** `( inmNmIndexType index, inmNmConditionType *condition )`
Parameter	
index	handle of related RX message
*condition	pointer to the status data
Return code	

Functional Description

This callback is executed within InmNmInit() and requests the user to initially configure the status data given by the pointer. It is up to the application to set the supervision counter and the supervision status.

This callback is meant to configure the supervision state with values that are e.g. stored in non-volatile memory.

Particularities and Limitations

- Call context: during system initialization; task level; while interrupts are locked
- This callback is only necessary if the user-specific initialization is enabled (see chapter “5 Configuration”).

ApplInmNmRxUserReInit()

Prototype	
Single channel	`void` **`ApplInmNmRxUserReInit`** `( inmNmIndexType index, inmNmConditionType *condition )`
Multi channel	`void` **`ApplInmNmRxUserReInit`** `( inmNmIndexType index, inmNmConditionType *condition )`
Parameter	
index	handle of related RX message
*condition	pointer to the status data
Return code	

Functional Description

This callback is executed within InmNmReInit() and requests the user to configure the status data given by the pointer. It is up to the application to set the supervision counter and the supervision status.

This callback is meant to configure the supervision state with values that are e.g. stored in non-volatile memory.

Particularities and Limitations

- Call context: during re-initialization; typically on task level
- This callback is only necessary if the user-specific initialization is enabled (see chapter “5 Configuration”).

8.3.3 TX Supervision

ApplInmNmStatusIndicationTx()

Prototype	
Single channel	`void` **`ApplInmNmStatusIndicationTx`** `( inmNmStatusType status )`
Multi channel	`void` **`ApplInmNmStatusIndicationTx`** `( inmNmChannelType inm_channel, inmNmStatusType status )`
Parameter	
inm_channel	Handle of the NM-internal channel. Range: 0…max(NM channels)
status	status of the TX supervision ■ INM_OK ■ INM_FAILURE ■ INM_CONFIRMED_FAILURE ■ INM_SPV_ACTIVE
Return code	

Functional Description	
This callback is executed when the status of the supervision for the channel-specific TX message changes, i.e. when a call of InmNmTxOk() or InmNmTxTimeOut() leads to a change of the supervision state. This callback is also executed during system initialization and re-initialization. The new status is contained in the parameter <status>.	
Particularities and Limitations	
■ Call context: maybe in IR context, depending on call context of InmNmTxOk() and InmNmTxTimeOut().	

ApplInmNmTxUserInit()

Prototype	
Single channel	`void ApplInmNmTxUserInit ( inmNmConditionType *condition )`
Multi channel	`void ApplInmNmTxUserInit ( inmNmChannelType inm_channel, inmNmConditionType *condition )`
Parameter	
inm_channel	Handle of the NM-internal channel. Range: 0…max(NM channels)
*condition	pointer to the status data
Return code	

Functional Description	
This callback is executed within InmNmInit() and requests the user to initially configure the status data given by the pointer. It is up to the application to set the supervision counter and the supervision status. This callback is meant to configure the supervision state with values that are e.g. stored in non-volatile memory.	
Particularities and Limitations	
■ Call context: during system initialization; task level; while interrupts are locked ■ This callback is only necessary if the user-specific initialization is enabled (see chapter “5 Configuration”).	

ApplInmNmTxUserReInit()

Prototype	
Single channel	`void ApplInmNmTxUserReInit ( inmNmConditionType *condition )`
Multi channel	`void ApplInmNmTxUserReInit ( inmNmChannelType inm_channel, inmNmConditionType *condition )`
Parameter	
inm_channel	Handle of the NM-internal channel. Range: 0…max(NM channels)
*condition	pointer to the status data
Return code	

Functional Description	
This callback is executed within InmNmReInit() and requests the user to configure the status data given by the pointer. It is up to the application to set the supervision counter and the supervision status. This callback is meant to configure the supervision state with values that are e.g. stored in non-volatile memory.	
Particularities and Limitations	
■ Call context: during re-initialization; typically on task level ■ This callback is only necessary if the user-specific initialization is enabled (see chapter “5 Configuration”).	

8.3.4 BusOff Supervision

ApplInmNmStatusIndicationBusOff()

Prototype	
Single channel	`void ApplInmNmStatusIndicationBusOff ( inmNmStatusType status )`
Multi channel	`void ApplInmNmStatusIndicationBusOff ( inmNmChannelType inm_channel, inmNmStatusType status )`
Parameter	
inm_channel	Handle of the NM-internal channel. Range: 0…max(NM channels)
status	status of the TX supervision ■ INM_OK ■ INM_FAILURE ■ INM_CONFIRMED_FAILURE ■ INM_SPV_ACTIVE
Return code	

Functional Description	
This callback is executed when the status of the channel-specific BusOff supervision changes, i.e. when a call of InmNmBusOff() or a notification of a TX/RX event leads to a change of the supervision state. This callback is also executed during system initialization and re-initialization. The new status is contained in the parameter <status>.	
Particularities and Limitations	
■ Call context: maybe in IR context, depending on the way CAN TX/RX/error interrupts are used in the system.	

ApplInmNmBusOffUserInit()

Prototype	
Single channel	`void ApplInmNmBusOffUserInit ( inmNmConditionType *condition )`
Multi channel	`void ApplInmNmBusOffUserInit ( inmNmChannelType inm_channel, inmNmConditionType *condition )`
Parameter	
inm_channel	Handle of the NM-internal channel. Range: 0…max(NM channels)
*condition	pointer to the status data
Return code	

Functional Description	
This callback is executed within InmNmInit() and requests the user to initially configure the status data given by the pointer. It is up to the application to set the supervision counter and the supervision status. This callback is meant to configure the supervision state with values that are e.g. stored in non-volatile memory.	
Particularities and Limitations	
■ Call context: during system initialization; task level; while interrupts are locked ■ This callback is only necessary if the user-specific initialization is enabled (see chapter “5 Configuration”).	

ApplInmNmBusOffUserReInit()

Prototype	
Single channel	`void ApplInmNmBusOffUserReInit ( inmNmConditionType *condition )`
Multi channel	`void ApplInmNmBusOffUserReInit ( inmNmChannelType inm_channel, inmNmConditionType *condition )`
Parameter	
inm_channel	Handle of the NM-internal channel. Range: 0…max(NM channels)
*condition	pointer to the status data
Return code	

Functional Description	
This callback is executed within InmNmReInit() and requests the user to configure the status data given by the pointer. It is up to the application to set the supervision counter and the supervision status. This callback is meant to configure the supervision state with values that are e.g. stored in non-volatile memory.	
Particularities and Limitations	
■ Call context: during re-initialization; typically on task level ■ This callback is only necessary if the user-specific initialization is enabled (see chapter “5 Configuration”).	

8.3.5 Generic Supervision

ApplInmNmStatusIndicationGeneric()

Prototype	
Single channel	`void` **`ApplInmNmStatusIndicationGeneric`** `( inmNmStatusType status )`
Multi channel	`void` **`ApplInmNmStatusIndicationGeneric`** `( inmNmChannelType inm_channel, inmNmStatusType status )`
Parameter	
inm_channel	Handle of the NM-internal channel. Range: 0…max(NM channels)
status	status of the TX supervision ■ INM_OK ■ INM_FAILURE ■ INM_CONFIRMED_FAILURE ■ INM_SPV_ACTIVE
Return code	

Functional Description	
This callback is executed when the status of the channel-specific Generic supervision changes, i.e. when a call of InmNmGenericOk() or InmNmGenericTimeOut() leads to a change of the supervision state. This callback is also executed during system initialization and re-initialization. The new status is contained in the parameter <status>.	
Particularities and Limitations	
■ Call context: maybe in IR context, depending on the call context of InmNmGenericOk() and InmNmGenericTimeOut()	

ApplInmNmGenericUserInit()

Prototype	
Single channel	`void ApplInmNmGenericUserInit ( inmNmConditionType *condition )`
Multi channel	`void ApplInmNmGenericUserInit ( inmNmChannelType inm_channel, inmNmConditionType *condition )`
Parameter	
inm_channel	Handle of the NM-internal channel. Range: 0…max(NM channels)
*condition	pointer to the status data
Return code	

Functional Description	
This callback is executed within InmNmInit() and requests the user to initially configure the status data given by the pointer. It is up to the application to set the supervision counter and the supervision status. This callback is meant to configure the supervision state with values that are e.g. stored in non-volatile memory.	
Particularities and Limitations	
■ Call context: during system initialization; task level; while interrupts are locked ■ This callback is only necessary if the user-specific initialization is enabled (see chapter “5 Configuration”).	

ApplInmNmGenericUserReInit()

Prototype	
Single channel	`void ApplInmNmGenericUserReInit ( inmNmConditionType *condition )`
Multi channel	`void ApplInmNmGenericUserReInit ( inmNmChannelType inm_channel, inmNmConditionType *condition )`
Parameter	
inm_channel	Handle of the NM-internal channel. Range: 0…max(NM channels)
*condition	pointer to the status data
Return code	

Functional Description	
This callback is executed within InmNmReInit() and requests the user to configure the status data given by the pointer. It is up to the application to set the supervision counter and the supervision status. This callback is meant to configure the supervision state with values that are e.g. stored in non-volatile memory.	
Particularities and Limitations	
■ Call context: during re-initialization; typically on task level ■ This callback is only necessary if the user-specific initialization is enabled (see chapter “5 Configuration”).	

8.4 Other Interfaces

8.4.1 Version Information

The version of the source code of the component Nm_IndOsek is stored in three BCD-coded constants within the NM source file:

```
V_MEMROM0 V_MEMROM1 vuint8 V_MEMROM2 kNmMainVersion   =
      (vuint8)(NM_INDOSEK_VERSION >> 8);
V_MEMROM0 V_MEMROM1 vuint8 V_MEMROM2 kNmSubVersion    =
      (vuint8)(NM_INDOSEK_VERSION & 0xFF);
V_MEMROM0 V_MEMROM1 vuint8 V_MEMROM2 kNmReleaseVersion =
      (vuint8)(NM_INDOSEK_RELEASE_VERSION);

Example - Version 1.24.03 is registered as:
kNmMainVersion    = 0x01;
kNmSubVersion     = 0x24;
kNmReleaseVersion = 0x03;
```

This information can be accessed by the application at any time.

9 Working with the Code

9.1 Version Information

The version information and the version changes of the NM component are listed in the history section at the beginning of the header and source code files.

9.2 Application Interface

The application uses the API of the NM. The necessary interfaces can be looked up in the NM header file. This file contains

- pre-processor definitions
- type definitions
- prototypes for API functions
- prototypes for necessary callback function

10 CANdb Attributes

The configuration tool needs additional information to handle the configuration options for the NM. These information are stored in attributes within the CAN database (DBC-file). Please refer to the online help system of the CANdb editor to learn how to define and set these attributes.

An attribute can belong to the database, to a node, to a message or to a signal.

There are different types of attributes: String, Hex, Integer-Values or Enumeration. When enumerations are used, please take care of the order (e.g. “No”, “Yes”).

The DBC attributes are normally provided by the OEM. The user does not have to change them.

Attribute Name	Valid for	Type	Value	Description
Manufacturer	Database	String	PSA, Renault	This attribute defines the OEM.
NmType	Database	String	PSA-Indirect-OSEK. Renault-Indirect-OSEK	This attribute defines the OEM-specific type of the NM.
NmNode	Node	Enumeration	“No”,”Yes”	This attribute defines if the corresponding node uses the NM (“Yes"") or not (“No”).
NmStationAddress	Node	Hex	range: 0x00…0xFF	This attribute defines the station address of the ECU. The station address has to be unique within the system.
NmMessage	Message	Enumeration	“No”,”Yes”	This attribute defines the TX message that will be used for TX supervision. Note: There may be only one TX message of each node which has this attribute set to “Yes”.
NmMessage_<node>	Message	Enumeration	“No”,”Yes”	This attribute defines the RX message(s) that will be used for RX supervision by node <node>. <node> represents the node name of the node of interest. This attribute may only be set to “Yes” for messages that are received by the own node.
GenMsgCycleTime	Message	Integer	1…2000	This attribute defines the cycle time for the supervised RX and TX messages. This value is used for the present detection of the supervised RX nodes, as well as for the absent/present detection of the supervised TX message. Note: The value range may not be exceeded for messages that are used for RX or TX supervision.
GenMsgTimeoutTime_<node>	Message	Integer	1..667	This attribute defines the timeout for the supervised RX messages. This value is used for the absent detection of the supervised RX nodes.

Table 10-1 CANdb attributes