# Nm_StMgrIndOsek_Ls

## Technical Reference

## Station Manager (Low Speed)

| | |
|---|---|
| **Version** | 3.05 |
| **Date** | 2011-07-29 |
| **File** | TechnicalReference_Stationmanager.doc |
| **Number of Pages** | 35 |

# 1 History

| Author | Date | Version | Remarks |
|---|---|---|---|
| Dieter Schaufelberger | 2008-01-29 | 3.00 | creation of this document |
| Dieter Schaufelberger | 2008-02-12 | 3.01 | Firest corrections |
| Dieter Schaufelberger | 2008-03-14 | 3.02 | Added new API, modify Sleep/WakeUp |
| Dieter Schaufelberger | 2008-06-25 | 3.03 | Minor corrections |
| Dieter Schaufelberger | 2008-07-11 | 3.04 | Added support of start delay time |
| Marco Pfalzgraf | 2011-07-28 | 3.05 | Update of user specification [INM PSA] |

## 1.1 SM Version

This document refers to version 3.03.00 of the station-manager for the PSA Low Speed Fault Tolerant bus.

# 2 Introduction

The aim of this document is to describe the handling of the station-manager for PSA.

This document contains

- a short description of the station-manager
- the condition for using the station-manager
- the interfaces of the user program for the station-manager

This chapter gives a brief overview of the tasks and aims of the station-manager.

Please refer also to the specification of the Indirect Network Management [INM PSA].

## 2.1 Reference Documents

For understanding and using this manual, it is very important to know the listed documents.

| Abbreviation Document | Document |
|---|---|
| [INM PSA] | 96 649 896 99 Ind[OR] Phases de vie reseau |
| | 96 649 897 9B Ind[OR] Specification de regles communication |
| [CANdriver] | User manual of the Vector CAN Driver |
| [INM_OSEK] | User manual of the Vector OSEK_INM |

## 2.2 Abbreviations

Instead of using complete expressions, the following abbreviations are used in the text.

| Abbreviation | Complete expression |
|---|---|
| ECU | Electronic Control Units |
| INM | Indirect Network Management |
| IL | Interaction Layer |

| | |
|---|---|
| SM | station-manager |

## 2.3 Tasks and Aims

The SM provides services for a user program operating on a CAN-Bus.

These services contain:

- Handling of network management states
- Handling of states of the supervised ECU's
- Handling of non volatile storage of error counters/states
- Monitoring and handling of "Perte_Com" situation (Limp Home)

# 3 Network management states

This chapter gives a brief description about the different ECU states and the ECU behaviour in those states according to [INM PSA].

## 3.1 Organ Type 1

Organ Type 1 is able to wake up the CAN communication in case of an external event (e.g. door control ). In case of Limp Home ( non reception of message "Commande_BSI") it enters state "Perte Com".

### 3.1.1 Description of internal States for organ type 1:

| State | Description |
|---|---|
| **VEILLE** | Physical Layer in Sleep mode. Detection of Bus activity.<br>No transmission possible. |
| **REVEIL** | Transient state in which the communication is initialised.<br>If the organ itself wakes up the bus a periodic wake up message must be send.<br>Supervision of the Commande_BSI. |
| **NORMAL** | Transmission of the wake up message is stopped.<br>Transmission of the version message.<br>Transmission and reception of functional messages.<br>Netmanagement diagnosis.<br>Supervision of the Commande_BSI. |
| **MIS EN VEILLE** | Transmission of functional messages interrupted.<br>Supervision of the Commande_BSI. |
| **COM OFF** | Transmission of functional messages interrupted.<br>Supervision of the Commande_BSI. |
| **Perte Com** | Transmission of functional messages. |

## 3.2 Organ Type 2

Organ Type 2 is able to wake up the CAN communication in case of an external event (e.g. door control ). In case of Limp Home ( non reception of message “Commande_BSI”) it enters state “MIS EN VEILLE”.

### 3.2.1 Description of internal States for organ type 2 :

| State | Description |
|---|---|
| **VEILLE** | Physical Layer in Sleep mode. Detection of Bus activity.<br>No transmission possible. |
| **REVEIL** | Transient state in which the communication is initialised.<br>If the organ itself wakes up the bus a periodic wake up message must be send.<br>Supervision of the Commande_BSI. |
| **NORMAL** | Transmission of the wake up message is stopped.<br>Transmission of the version message.<br>Transmission and reception of functional messages.<br>Netmanagement diagnosis.<br>Supervision of the Commande_BSI. |
| **MIS EN VEILLE** | Transmission of functional messages interrupted.<br>Supervision of the Commande_BSI. |
| **COM OFF** | Transmission of functional messages interrupted.<br>Supervision of the Commande_BSI. |

## 3.3 Organ Type 3

ECUs Organ Type 3 are supplied by +CAN which is switched off by the BSI when going into the state VEILLE. Therefore these ECUs aren’t able to wake up the CAN communication. In case of Limp Home ( non reception of message “Commande_BSI”) it enters state “Perte Com”.

### 3.3.1 Description of internal States for organ type 3 :

| State | Description |
|---|---|
| **VEILLE** | +CAN absent => no power supply<br>Detection of Bus activity not possible.<br>Transmission of any message impossible. |
| **REVEIL** | Transient state in which the communication is initialised.<br>Supervision of the Commande_BSI. |
| **NORMAL** | Transmission of the wake up message is stopped.<br>Transmission of the version message.<br>Transmission and reception of functional messages.<br>Netmanagement diagnosis.<br>Supervision of the Commande_BSI. |
| **MIS EN VEILLE** | Transmission of functional messages interrupted.<br>Supervision of the Commande_BSI. |
| **COM OFF** | Transmission of functional messages interrupted.<br>Supervision of the Commande_BSI. |

| | |
|---|---|
| **Perte Com** | Transmission of functional messages. |

## 3.4 Organ Type 4

Organ Type 4 is able to wake up the CAN communication in case of an external event (e.g. door control ). In case of Limp Home ( non reception of message “Commande_BSI”) it enters state “MIS EN VEILLE”.

### 3.4.1 Description of internal States for organ type 4 :

| State | Description |
|---|---|
| **VEILLE** | Physical Layer in Sleep mode. Detection of Bus activity.<br>No transmission possible. |
| **REVEIL** | Transient state in which the communication is initialised.<br>Supervision of the Commande_BSI. |
| **NORMAL** | Transmission of the wake up message is stopped.<br>Transmission of the version message.<br>Transmission and reception of functional messages.<br>Netmanagement diagnosis.<br>Supervision of the Commande_BSI. |
| **MIS EN VEILLE** | Transmission of functional messages interrupted.<br>Supervision of the Commande_BSI. |
| **COM OFF** | Transmission of functional messages interrupted.<br>Supervision of the Commande_BSI. |

# 4 Integration into the application

This chapter describes the steps for the integration of the SM in the application of an ECU.

## 4.1 Delivery Items

The SM is always combined with the indirect network management OSEK_INM. Nevertheless it is an own module which could be used stand-alone ( not recommended ).

The delivery includes

- Header files *Stat_mgr.h*
- Source code file (includes the SM itself) *Stat_mgr.c*
- Source code file *GenericPrecopy.c*

this user manual *Technical Reference station-manager.*

## 4.2 Version Changes

Changes and bug fixes in the SM are listed at the beginning of the header and source code file.

## 4.3 Handling of the station-manager

Please include the header file of the SM in all modules in which you require services of the SM. In this file all available services including prototypes of the required interfaces and symbolic constants are defined.

Add *Stat_mgr.c* , *GenericPrecopy.c* and the generated file *StMgrLs_par.c* to your make file/project file.

Before the SM can be used, it must be initialised once after power on reset. Therefore the function *SmInitPowerOn()* has to be called.

After the SM was initialized, the function *SmTask(…)* must be called cyclically with the period time defined in the Generation tool.

All other services of the SM have to be called by the application when they are required. A more detailed description of every function is found in the chapter "API of the station-manager".

## 4.4 Particularities if there is no Interaction Layer in the system

If there is no Interaction Layer present in the system, the Station manager takes care about the cyclic sending of the supervised Tx message.

### 4.4.1 Event transmission of the supervised Tx message

On some systems this message has to be send also on event. Therefore an interface is provided to handle this:

**SmTransmitNmMessage: Tranmsit the supervised Tx message on next call of SmTask()**

| **Prototype** | **Void SmTransmitNmMessage( channel)** |
|---|---|
| **Parameter** | Channel Channel on which the supervised message must be sent |
| **Return code** | |
| **Function de-scription** | This function checks for the minimum send delay and sets the internal Tx cycle counter to the appropriate value to transmit the supervised Tx message as soon as possible (keeping the minimum send delay). |
| **Particularities and Limitations** | |

## 4.5 Start delay time of Tx messages

If the Tx messages have to keep a defined start delay time, the database attribute GenMsgStartDelayTime has to be set for each Tx message to the corresponding delay time.

### 4.5.1 Start delay time without Interaction Layer

If there is no Interaction Layer in the system, the start delay time has to be kept by the application.

For the supervised Tx messages, which is transmitted by the Station manager (s. chap. 4.4), the start delay time has to be provided by the application using the following array:

```
V_MEMROM0 V_MEMROM1 vuint16 V_MEMROM2 bSmInmMsgDelayTime[SM_CHANNELS]= {x};
```

The delay time "x" has to be given normalized, which means "Delay Time / Task cycle".

A delay time of 25ms using a task cycle of the Stationmanager of 5ms gives x= 25/5 = 5.

```
V_MEMROM0 V_MEMROM1 vuint16 V_MEMROM2 bSmInmMsgDelayTime[SM_CHANNELS]= {5};
```

## 4.6 Reading of Nerr state

The state of the Bit Nerr is read by the station-manager using the callback function *ApplCanErrorPin().* In this callback function the application has to read the Nerr pin of the transceiver. According to the requirement GEN-RESEAU-ST-RCCANLS.0095 (1) the state of the Bit Nerr must be read maximum 88µs after the reception of the Commande_BSI. This requirement can not be met under all circumstances. Depending on the system design (platform, hardware clock, polling mode, … ) this time may be exceeded. Therefore this timing ** must ** be measured using the real target on customer side.

If the requirement isn't met, the following solution has to be implemented:

Activate callback function ApplCanMsgReceived in the CanDriver folder of the Gentool. In this callback function the application has to read the state of the Nerr pin and store it in a variable. If later on the callback function ApplCanErrorPin() is called, return the stored value.

This will ensure that the Nerr pin is read immediately after the reception of the message.

Disadvantage:

State of Nerr pin is read with every message not only Commande_BSI.

## 4.7 Handling of the signal Interd_Memo_Def

To activate the handling of the signal Interd_Memo_Def select the switch *InterdMemoDef support* in the Generation tool.

If the handling is activated, the application has to provide the callback function **ApplSmGetInterdMemoDef( )** which return 0 if the signal is not set and >0 if the signal is set.

If the signal is set, the SM keeps all volatile counters unchanged ( absent, mute, nerr,) when in mode **Normal** or **ComOff**.

The PerteCom supervision is still active.

## 4.8 Handling of Lmin

If there are messages where the Lmin value is smaller as the specified DLC, special care has to be taken when reading signals of those messages. Lmin gives the minimal length of a message until its still accepted by the ECU. Lmin can be equal to message DLC or smaller. To ensure the correct functionality of the stack please set the database message attribute **GenMsgMinAcceptLength** to Lmin or DLC if there is no Lmin value given ( s. 6.1 )

For messages with Lmin < DLC special care has to be taken by the application. First of all a short description how the available signal flags are handled by the IL.

### 4.8.1 First value and Indication flags

If the received DLC is >= Lmin the **firstvalue, and indication** flags are only set for the signals which are entirely contained in the message. Also the **indication** function is only called for these signals.

### 4.8.2 Timeout flags and functions

If the message is missing but was received before with a DLC > Lmin, **timeout** flags and functions are only called for the signals entirely contained in the DLC received last time. If the message was never received, the flags and functions are set and called for all signals.

### 4.8.3 Impact on the application

Before accessing a signal the application has to prove if the signal was received by checking the corresponding indication flag. If the indication flag is set, the signal was entirely received and can be used. Otherwise the default value must be used. For the further handling two cases must be treated:

**1. The DLC will not change during a session ( Init -> Stop )**

This means the message is always received with the same DLCmessage where Lmin <= DLCmessage <= DLC. In this case the indication flags are set once and don't need to be cleared by the application. The signal can be accessed whenever needed (of course the indication flag still has to be checked !).

**2. The DLC may change with every message**

This means the message may have a different DLC on every reception. In this case the application must clear the indication flags when a new message is received before they are set by the IL. This ensures that the indication flags are set correctly on every reception and no invalid signals are read by the application.

To do so the application has to define a message precopy function for this message where all the indication flags are cleared. Additionally IL polling must be inactive for this message.

**Example:**

| | |
|---|---|
| Message name : | message_1 |
| Precopy function: | message_1_precopy (the name is assigned in the generation tool ) |
| Signal names: | signal_1, signal_2, signal_3 |

```
Data Access in the application
***********************************************
/* DATA Access signal_1*/
if( ILGetsignal_1Ind() ) /* Signal valid? */
{
/* Access of the data via IL macro */
      variable_1 = ILGetRxsignal_1();
}
else
{
      variable_1 = <Default> /* Default value */
}
***********************************************
/* DATA Access signal_2*/
if( ILGetsignal_2Ind() ) /* Signal valid? */
{
/* Access of the data via IL macro */
```

```
      variable_2 = ILGetRxsignal_2();
}
else
{
      variable_2 = <Default> /* Default value */
}
***********************************************
/* DATA Access signal_3*/
if( ILGetsignal_1Ind() ) /* Signal valid? */
{
/* Access of the data via IL macro */
      variable_3 = ILGetRxsignal_3();
}
else
{
      variable_3 = <Default> /* Default value */
}
***********************************************
```

**message precopy function**

```
***********************************************
vuint8 message_1_precopy(…)
{
/* clear all indication flags via IL macros*/
      ILClrsignal_1Ind();
      ILClrsignal_2Ind();
      ILClrsignal_3Ind();
      return kCanCopyData;
}
***********************************************
```

### 4.8.4 Access to the actual DLC

The actual received DLC of a message is written to the variable

*canVariableRxDataLen[index]* where index is the receive handle of the message. This message handles are defined in the <board1.h> file which is generated by the Generation tool. In the example above the handle would be defined as *CanRxmessage_1*

```
/* get the actual DLC */

actual_dlc = canVariableRxDataLen[CanRxmessage_1];
```

## 4.9 Cancel of pending transmit messages

For PSA the application has to cancel a transmit message as soon as there is no confirmaition for 15ms-20ms (N_as time).

Depending on the configuration, the supervision of the N_as time and the removing of the messages from the transmit buffer can be handled by the station-manger. In this case also the actual value of the N_as counter is written to the corresponding signal.

Prerequisite is a call of the SmTask() function in a cycle time not greater 5ms. This is due to the granularity of the counter and must be kept to fulfil the timing constraint of 15-20 ms.

The TxObserve functionality must be activated in the CAN driver configuration.

Additionally a special Attribute must be set in the database for the signal containing the value of the N_as counter:

Attribute:

| | |
|---|---|
| **Name** | GenSigSpecialFunction |

| | |
|---|---|
| **Type** | String |
| **Elements** | - |
| **Value** | NmOsekI_Nas ( attention: letter after Osek is <u>not</u> minor **L** but major **i** ) |
| **Particularities** | This attribute has to be set for the signal containing the value of the N_as counter to be transmitted to the BSI ECU |

**Note:**

Also if N_as supervision is performed by the station-manager chapter 4.9.1 has to be taken into account.

If SmTask() can’t be called in a cycle time of less or equal 5ms, the supervision of the N_as time has to be done by the application. To do so, you have to activate the TxObserve functionality in the GenTool-> CAN Driver page. Every time a message is copied into the transmit buffer the Callback funtion

**void ApplCanTxObjStart ( CanObjectHandle txHwObject )**

is called. Now a timer can be started. If the message was transmitted successful, the callback function

**void ApplCanTxObjConfirmed ( CanObjectHandle txHwObject )**

is called and the corresponding timer is cancelled.

If the timer isn’t cancelled in time,

**void CanCancelTransmit ( txHandle )**

has to be called. To get the txHandle from the txHwObject the function

**txHandle = CanTxGetActHandle(CanObjectHandle txHwObject )**

is provided. Since in most systems the transmission is running in interrupt context whereas the application runs in task context, it is possible that while the timeout N_as is detected the message is transmitted. Therefore the value returned from this function **must** be checked by the application to avoid calling CanCanceltTransmit (..) with a invalid txHandle.

```
If ( txHandle < kCanNumberOfTxObjects )
{
    /* txHandle is valid */
    Message was not sent, increment N_as counter
}
else
{
    /* txHandle invalid */
    Message was sent, don’t increment N_as counter
}
```

**Note:**

The return value must be checked! Otherwise data may be overwritten by the function CanCancelTransmit(..).

In case the Can driver is reinitialised, the callback function

**void ApplCanInit (CAN_CHANNEL_CANTYPE_FIRST CanObjectHandle txHwObjectFirstUsed, CanObjectHandle txHwObjectFirstUnused )**

is called. In this case all pending N_as timers must be stopped.

After the cancellation of a transmit message, the transmit object of the CAN driver is not released automatically because dependent on the hardware platform, internal states have to be checked. To release the transmit object the function **CanTxTask()** or **CanTask()** must be called by the application **periodically**. The period time should be as short as possible as the transmit object will be blocked until it is released by ***CanTxTask()***.

**Note:**

Since the transmit object is only released if internal states are met, it is not sure that it is released with the first call of **CanTxTask()**. Therefore it is not sufficient to call it once after you called **CanCancelTransmit()**. It must be called periodically!

### 4.9.1 Configuration of the TP when using CanCanelTransmit()

If the N_as time is supervised by the application special care has to be taken when configuring the TP. The TP also provides a supervision of the N_as time for the TP messages. These messages are now also supervised by the application. To avoid unexpected behaviours caused by interferences of TP and application, make sure that the N_as time configured in the TP is at least double the time of the N_as time supervised by the application. In case of PSA this means 2*15ms = 30ms.

### 4.9.2 Example Code

An example code how to implement the N_as supervision is given in the file demo_nas.c which is part of the delivery.

## 4.10 Handling of the version message

The version message is transmitted by the station-manger in case of a state change from state *Reveil to Normal.* The version message must be indicated to the Stationmanager by setting the signal attribute *GenSigSpecialFunction* in the database to *NmOsekI_Version* for one of the signals contained in the version message.

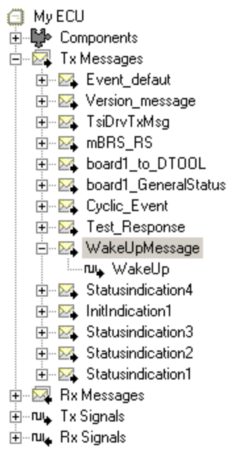
## 4.11 Configuring the Part Offline mode

To provide a correct handling of the transmit messages the user has to set up the part offline modes for the transmit messages. Using the Part Offline functionality of he CAN driver, the station manager controls the transmit permissions for the transmit messages. E.g. the wake up message must only be transmitted in state wake up ( reveil ).

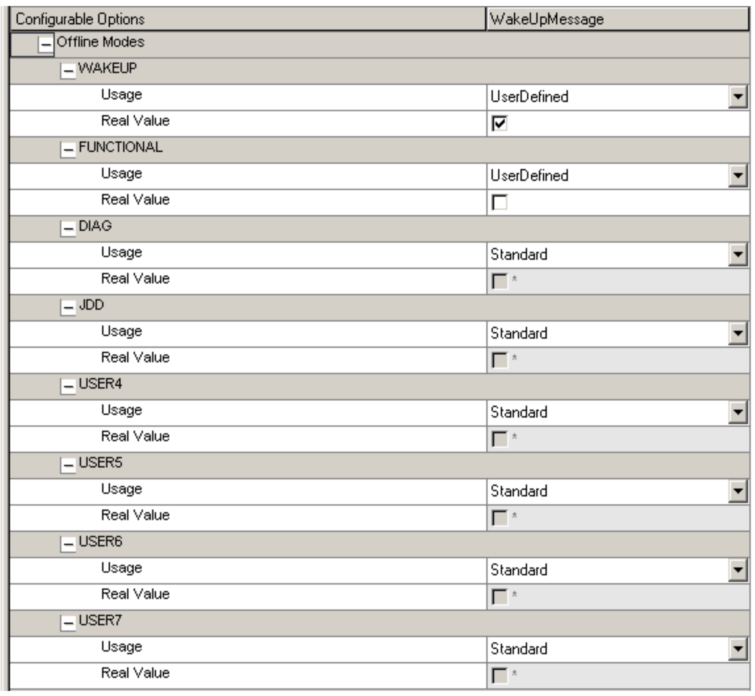
There are 4 different Offline groups: FUNCTIONAL, WAKEUP, JDD and DIAG.

By assigning the corresponding messages to these 4 groups they are only transmitted in the state they are allowed. The wakeup message to the WAKEUP group, the functional messages to the FUNCTIONAL group, the JDD message to the JDD group and the diagnosis messages to the DIAG group.

The configuration of the offline groups is done in the TxMessages section on each message ( s.below an example for the WakeUpMessage ).



```
My ECU
  Components
  Tx Messages
    Event_defaut
    Version_message
    TsiDrvTxMsg
    mBRS_RS
    board1_to_DTOOL
    board1_GeneralStatus
    Cyclic_Event
    Test_Response
    WakeUpMessage
      WakeUp
    Statusindication4
    InitIndication1
    Statusindication3
    Statusindication2
    Statusindication1
  Rx Messages
  Tx Signals
  Rx Signals
```



| Configurable Options | WakeUpMessage |
|---|---|
| Offline Modes | |
| WAKEUP | |
| Usage | UserDefined |
| Real Value | ☑ |
| FUNCTIONAL | |
| Usage | UserDefined |
| Real Value | ☐ |
| DIAG | |
| Usage | Standard |
| Real Value | ☐ * |
| JDD | |
| Usage | Standard |
| Real Value | ☐ * |
| USER4 | |
| Usage | Standard |
| Real Value | ☐ * |
| USER5 | |
| Usage | Standard |
| Real Value | ☐ * |
| USER6 | |
| Usage | Standard |
| Real Value | ☐ * |
| USER7 | |
| Usage | Standard |
| Real Value | ☐ * |

## 4.12 Supervision Reset on Request of the Diagnosis

On request of the diagonsis, the network supervision must be reset. Therefore a specific API is provided which performs the reset of counters and states of the supervision:

**SmResetSupervision();**

When this API is called the Supervision is reset to the follwing values:

| | **Volatiler counters** | **State** | **Supervison** |
|---|---|---|---|
| BSI **not** confirmed absent | 0 | Confirmed OK | Active/deactive (as before reset) |
| BSI confirmed absent | 0 | Confirmed OK | Active |

# 5 Sleep and Wake Up sequence PSA

## 5.1 Transition from Sleep ( Veille ) to WakeUp ( Reveil )

After a nominal transition to sleep or after the initialization of the station-manager, it is in state *sleep*. Depending on the Sleep management type (Organ type) configured in the Generation tool, there are 1-3 events which will start the wake up sequence of the ECU:

### 5.1.1 External event ( Organ type 1, 2 and 4 ):

In case of an external event, the application has a request to wake up the network. To indicate the need of the network communication the API function **SmSetNetworkRequest*()*** is called. Within the next call of SmTask(), SM and INM are started and the callback function **ApplSmTrcvOn()** is called. This callback can be used to wake up the CAN driver and start the IL ( if present ):

```
ApplSmTrcvOn()
{…
  CanWakeUp();
  IlTxStart();
  IlRxStart();
  …
}
```

The SM enters state wakeup (reveil) and starts to transmit the Wakeup message.

***Note:*** *As long as the NetworkRequest is active, the Station-manager will not enter sleep mode. Due to this it is required that the application releases the network request if network communication is not required anymore. This can be done by the API* ***SmReleaseNetworkRequest()***

*If the Request isn't released, the Nm will try to wake up the network as soon as "MODE NORMAL" is left.* ***This function isn't a trigger to have a single WakeUp event! If set the Nm will try to wake Up the network until the maximum number of allowed attempts is exceeded!***

*If a single Wake Up is required, the Request must be released using SmReleaseNetworkRequest() as soon as the Wake up attempt failed ( no state change to normal )*

***Note:*** *The SM only takes control about offline groups (thus if the message is allowed to be sent in this ECU state or not ). If there is an IL in the system and the Wakeup message is configured as cyclic message, the application doesn't have to take care about the transmission of it.*

If there is no IL in the system, the application has to take care about the transmission and the transmit cycle of the Wakeup message.

### 5.1.2 Bus event ( Organ type 1, 2 and 4 )

In case of a Bus event ( reception of a CAN message ) the callback ***ApplCanWakeUp()*** is called in the application, which indicates the wake up interrupt. To start up the networmanagement the API **SmSetWakeUpRequest*()*** has to be called.

Within the next call of SmTask(), SM and INM are started and the callback function **ApplSmTrcvOn()** is called. This callback can be used to wake up the CAN driver and start the IL ( if present ):

```
ApplSmTrcvOn()
{…
  CanWakeUp();
  IlTxStart();
  IlRxStart();
  …
}
```

The SM enters state wakeup (reveil) and starts to transmit the Wakeup message.

***Note:*** *even so the CAN cell is already awake, the call of* ***CanWakeUp()*** *doesn't cause a malfunction.*

In case of  Bus event, no wake up message is transmitted.

### 5.1.3 +CAN activation Organ type 1 and 2

In case of a +CAN event ( +CAN line changes to active ), the new state of the +CAN line has to be indicated to the network management by using the API ***SmSetPlusCan-State(kSmPlusCanActive)***. If in state sleep, the wake up sequence is started and the callback function **ApplSmTrcvOn()** is called. This callback can be used to wake up the CAN driver and start the IL ( if present ):

```
ApplSmTrcvOn()
{…
  CanWakeUp();
  IlTxStart();
  IlRxStart();
  …
}
```

The SM enters state wakeup (reveil) and starts to transmit the Wakeup message.

No wake up message must be transmitted in this case.

### 5.1.4 +CAN activation Organ type 3

For organ type 3 two different situations have to be distinguished:

**ECU is powered by the +CAN line only**

In case the ECU is only powered by the +CAN line, the ECU is powered up as soon as +CAN line is present. After initialization the application has to call the API ***SmSetPlus-CanState(kSmPlusCanActive)*** to put the SM into state "Reveil". The callback function **ApplSmTrcvOn()** is called and can be used to start the IL ( if present ):

```
ApplSmTrcvOn()
{…
  IlTxStart();
  IlRxStart();
  …
}
```

The CAN driver don't have to be started in this use case.

**ECU is powered by an additional power supply**

If the ECU is powered by a different power supply as +CAN, the +CAN behaviour has to be simulated. This means that during the absence of +CAN the ECU must not acknowledge any CAN frame. Therefore the CAN driver must be stopped when +CAN is absent. For the wake up sequence the application has to supervise the +CAN signal. If +CAN is switched on, the API ***SmSetPlusCanState(kSmPlusCanActive)*** has to be called. The callback function **ApplSmTrcvOn()** is called and can be used to start the CAN driver and IL ( if present ):

```
ApplSmTrcvOn()
{…
  CanStart();
  IlTxStart();
  IlRxStart();
  …
}
```

In this case the CAN driver must be started with the API ***CanStart()*** as it is in stop mode for organ type 3 with additional power supply.

## 5.2 Transition to Veille

### 5.2.1 Organ type 1,2 and 4

Whenever there is a request (either by BSI or the netmanagement ) to switch to the state *VEILLE* , the SM waits for one seoncd ( as specified ) and enters the state “Veille”. The INM is stopped and the callback function **ApplSmTrcvOff()** is called. This callback can be used to stop the IL ( if present ) and put the CAN driver into sleep mode:

```
ApplSmTrcvOff()
{…
  IlTxStop();
  IlRxStop();
  CanSleep();
  …
}
```

The required calls to **CanOffline()** and **CanResetBusSleep()** (s. Driver manual ) are performed by the SM already.

### 5.2.2 Organ type 3

**ECU is powered by the +CAN line only**

In this case no action is required by the application as the power supply is switched off during transition to state “Veille” by the the BSI ECU.

**ECU is powered by the +CAN line only**

If the ECU is powered by an additional power supply, the application has to call the API ***SmSetPlusCanState(kSmPlusCanDeactive)***. This will stop the SM and INM immediately and the callback function **ApplSmTrcvOff()** called. This callback can be used to stop the IL ( if present ) and put the CAN driver into stop mode:

```
ApplSmTrcvOff()
{…
  IlTxStop();
  IlRxStop();
  CanStop ();
  …
}
```

***Note:** In this use case the CAN driver must be put into **stop mode** as to avoid that the CAN driver is woken up by a frame during a deactivated +CAN line.*

## 5.3 Setting and Releasing a Request for the network

To avoid that the ECU enters sleep mode because the application needs the network, the network must set a request to the network management. On the other hand side if the application is able to sleep ( doesn't need the network at the moment ) the application **must** release the request to the network management as well.

***If a request isn't released, the ECU will never enter the sleep mode!***

To set and release a request two macros are provided :

**SmSetWakeUpRequest( )** to set

**SmReleaseWakeUpRequest( )** to release

## 5.4 Indication of the +CAN signal to the Station manager

As the internal state transition of the SM depends also on the +CAN signal, the state of the +CAN line has to be indicated to the SM whenever the state changes. To indicate the state the API ***SmSetPlusCanState(state)*** is provided.

In case of organ type 3 with additional power supply, calling this API with state *kSmPlus-CanDeactive* will put the SM into Sleep mode immediately.

# 6 API of the station-manager

The SM application program interface consists of services, which are realised by function calls. The services are called wherever they are required. They transfer information to the SM or take over information from the SM. This information is saved in the SM until it is not required anymore, respectively until it is changed by other operations.

## 6.1 Version of the source code

The source code version of the SM is provided by three BDC coded constants:

kStatMgrMainVersion = (uint8)((STATION_MANAGER_VERSION >> 8) );

kStatMgrSubVersion = (uint8)(STATION_MANAGER_VERSION );

kStatMgrBugFixVersion = (uint8)(STATION_MANAGER_BUGFIX_VERSION);

Example :

Version 3.20 is registered as (Bugfix Version 0):

kStatMgrMainVersion = 0x03;

kStatMgrSubVersion = 0x20;

kStatMgrBugfixVersion = 0x00;

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

## 6.2 station-manager services called by the application

### 6.2.1 SmInitPowerOn: Initialisation of the station-manager

| | |
|---|---|
| **Prototype** | void **SmInitPowerOn** ( void ) |
| **Parameter** | |
| **Return code** | |
| **Functional description** | Both the indirect OSEK network management and the station manager are initialized.<br>- variables are set to default values<br>- internal/external state: “VEILLE”<br>- reset BusOff recovery algorithm<br>- initialise NM |
| **Particularities and Limitations** | The CAN interrupts (Tx, Rx, Error) have to be disabled. |

### 6.2.2 SmTask: cyclic Task

| | |
|---|---|
| **Prototype** | void **SmTask**(void / channel ) |
| **Parameter** | Channel If multichannel system |
| **Return code** | |
| **Functional description** | The complete polling handling is done in this function:<br>- network state transitions by slave/master ECU<br>- diagnostic handling for (non-)volatile counters<br>- Tx and Rx timeout observation<br>- different BusOff recovery algorithms (MEDIUM, SLOW)<br>- wakeup request to force state “MODE NORMAL”<br>- sending of network management message |
| **Particularities and Limitations** | The SM must be initialised when this function is called. |

### 6.2.3 SmGetStatus: Read the internal status of ECU

| | |
|---|---|
| **Prototype** | vuint 8 **SmGetStatus**(void / channel) |
| **Parameter** | Channel If multichannel system |
| **Return code** | Internal Status of ECU (variable “bSmIntState “) |
| **Functional description** | This function returns the current status of the ECU (e.g. to suppress the timeout observation. |
| **Particularities and Limitations** | The SM must be initialised when this function is called. |

### 6.2.4 SmGetVolCNerr: Read the Value of the volatile Nerr Counter (Macro)

| | |
|---|---|
| **Prototype** | **SmGetVolCNerr(**void / channel**)** |
| **Parameter** | Channel If multichannel system |
| **Return code** | |
| **Macro description** | This macro returns the current value of the volatile Nerr counter |
| **Particularities and Limitations** | |

## 6.2.5 SmSetVolCNerr: Set the Value of the volatile Nerr Counter (Macro)

| | | |
|---|---|---|
| **Prototype** | **SmSetVolCNerr (**void / channel **,** val **)** | |
| **Parameter** | Val | Vuint8 Value to be set |
| | Channel | If multichannel system |
| **Return code** | | |
| **Macro description** | This macro sets the current value of the volatile Nerr counter | |
| **Particularities and Limitations** | | |

## 6.2.6 SmGetVolCPerteCom: Read the Value of the volatile PerteCom Counter (Macro)

| | |
|---|---|
| **Prototype** | **SmGetVolCPerteCom(**void / channel**)** |
| **Parameter** | Channel If multichannel system |
| **Return code** | |
| **Macro description** | This macro returns the current value of the volatile PerteCom counter |
| **Particularities and Limitations** | |

## 6.2.7 SmSetVolCPerteCom: Set the Value of the volatile PerteCom Counter (Macro)

| | | |
|---|---|---|
| **Prototype** | **SmSetVolCPerteCom (**void / channel **,** val**)** | |
| **Parameter** | Val | Vuint8 Value to be set |
| | Channel | If multichannel system |
| **Return code** | | |
| **Macro description** | This macro sets the current value of the volatile PerteCom counter | |
| **Particularities and Limitations** | | |

## 6.2.8 SmGetVolCBoff: Read the Value of the volatile BusOff Counter (Macro)

| | |
|---|---|
| **Prototype** | **SmGetVolCBoff (**void / channel**)** |
| **Parameter** | Channel If multichannel system |
| **Return code** | |
| **Macro descrip-** | This macro returns the current value of the volatile BusOff counter |

| | |
|---|---|
| **tion** | |
| **Particularities and Limitations** | |

### 6.2.9 SmSetVolCBoff: Set the Value of the volatile BusOff Counter (Macro)

| | | |
|---|---|---|
| **Prototype** | **SmSetVolCBoff (**void / channel **,** val**)** | |
| **Parameter** | Val | Vuint8 Value to be set |
| | Channel | If multichannel system |
| **Return code** | | |
| **Macro description** | This macro sets the value of the volatile BusOff counter | |
| **Particularities and Limitations** | | |

### 6.2.10 SmSetWakeUpRequest: A CAN frame was received and woke up the ECU

| | | |
|---|---|---|
| **Prototype** | Void **SmSetWakeUpRequest(**void / channel**)** | |
| **Parameter** | Channel | If multichannel system |
| **Return code** | | |
| **Functional description** | A CAN frame was received by the ECU and the NM must be started. | |
| **Particularities and Limitations** | This function is only available for Organtyp 1, 2 and 4 | |

### 6.2.11 SmSetNetworkRequest: Release a Network Request to the network management

| | | |
|---|---|---|
| **Prototype** | **SmSetNetworkRequest(**void / channel**)** | |
| **Parameter** | Channel | If multichannel system |
| **Return code** | | |
| **Functional description** | The application needs the CAN network. Therefore a network request is set. If the ECU isn't in application mode a specific wakeup message is sent periodically to force the master ECU to change the state of all slave ECUs to "MODE NORMAL". | |
| **Particularities and Limitations** | This macro is only available for Organtyp 1, 2 and 4.<br><br>If the Request isn't released, the Nm will try to wake up the network as soon as "MODE NORMAL" is left. **This function isn't a trigger to have a single WakeUp event! If set the Nm will try to wake Up the network until the maximum number of allowed attempts is exceeded!**<br><br>If a single Wake Up is required, the Request mus be released using *SmReleaseNetworkRequest()* as soon as the Wake up attempt failed ( no state change to normal ) | |

### 6.2.12 SmReleaseNetworkRequest: Release a Network Request to the network management

| | | |
|---|---|---|
| **Prototype** | **SmReleaseNetworkRequest(**void / channel**)** | |
| **Parameter** | Channel | If multichannel system |
| **Return code** | | |

| | |
|---|---|
| **Functional description** | The application doesn't need the CAN network anymore. Therefore the network request is released and the ECU can enter Sleep mode |
| **Particularities and Limitations** | This macro is only available for Organtyp 1, 2 and 4. |

### 6.2.13 SmSetPlusCanState(state)

| | | |
|---|---|---|
| **Prototype** | **SmSetPlusCanState(**void / channel, state **)** | |
| **Parameter** | State (vuint8) | State of the +CAN signal: 1 +CAN active; 0 +CAN not active |
| | Channel | If multichannel system |
| **Return code** | | |
| **Functional description** | The application indicates the state of the +CAN line to the station manager<br>This function is also called to wake up the ECU in case of a +CAN Wake Up event. | |
| **Particularities and Limitations** | This macro is only available for Organtyp 1+ 2 and 3. | |

### 6.2.14 SmTransmitNmMessage: Tranmsit the supervised Tx message on next call of SmTask()

| | | |
|---|---|---|
| **Prototype** | **SmTransmitNmMessage(**void / channel **)** | |
| **Parameter** | Channel | Channel on which the supervised message must be sent |
| **Return code** | | |
| **Macro description** | This macro sets the internal Tx cycle counter to 1, so on the next call of SmTask() the supervised Tx message is sent. | |
| **Particularities and Limitations** | | |

## 6.3 Application functions required by the station-manager

The prototypes of the functions required by the SM are defined in the header.

**Note:**

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

*These functions are not allowed to change the interrupt status. This is expected by the corresponding functions of the SM.*

### 6.3.1 ApplSmStatusIndicationTx: Status of transmission indication

| | |
|---|---|
| **Prototype** | void **ApplSmStatusIndicationTx**( inmNmStatusType status) |
| **Parameter** | status Status of transmission. |
| **Return code** | |
| **Functional description** | User specific function which is called if the state of transmission will be changed. The state which will be entered is passed as parameter. |
| **Particularities and Limitations** | This function will only be called if configured. Please refer to the chapter about configuration. |

### 6.3.2 ApplSmStatusIndicationRx: Status of reception indication

| | |
|---|---|
| **Prototype** | void **ApplSmStatusIndicationRx**( inmNmIndexType index, inmNmStatusType status ) |
| **Parameter** | index Handle of the monitored receive message. Handle can be 0 to r-1 (r = number of all monitored receive messages) starting with 0.<br>status Status of reception. |
| **Return code** | |
| **Functional description** | User specific function which is called if the state of reception will be changed. The state which will be entered is passed as parameter. |

### 6.3.3 ApplSmStatusIndicationNerr: Status of Nerr Pin indication

| | |
|---|---|
| **Prototype** | void **ApplSmStatusIndicationNerr**( inmNmStatusType status ) |
| **Parameter** | status Status of reception. |
| **Return code** | |
| **Functional description** | User specific function which is called if the state of Nerr will be changed. The state which will be entered is passed as parameter. |
| **Particularities and Limitations** | This function will only be called if configured. Please refer to the chapter about configuration. |

### 6.3.4 ApplSmStatusIndication: State of ECU has changed

| | |
|---|---|
| **Prototype** | void **ApplSmStatusIndication**( inmNmStatusType status) |
| **Parameter** | status Status of transmission. |
| **Return code** | |
| **Functional description** | User specific function which is called if the state of transmission will be changed. The state which will be entered is passed as parameter. |
| **Particularities and Limitations** | This function will only be called if configured. Please refer to the chapter about configu-ration. |

### 6.3.5 ApplCanErrorPin: Get Status of Transceiver Error Pin

| | |
|---|---|
| **Prototype** | canuint8 **ApplCanErrorPin**( void ) |
| **Parameter** | |
| **Return code** | Status of Error Pin. Encoding: 0 no Error , 1 Error |
| **Functional description** | User specific function which is called when the SmCanNerrAccess ( ) function is exe-cuted. It returns the Status of the Error Pin of the transceiver, if available. |
| **Particularities and Limitations** | This function will only be called if configured. Please refer to the chapter about configu-ration. |

### 6.3.6 ApplSmGetInterdMemoDef: Get Status of filtered Interd_Memo_Def Bit

| | |
|---|---|
| **Prototype** | canuint8 **ApplSmGetInterdMemoDef**( void ) |
| **Parameter** | |
| **Return code** | Filtered state of the Interd_Memo_Def signal ( 0 signal is not set , >0 signal is present ) |
| **Functional description** | This function returns the filtered state of the signal Interd_Memo_Def. While this bit is set, the values of the volatile counters aren't changed. If this signal isn't provided, the function must return 0. |
| **Particularities and Limitations** | |

### 6.3.7 ApplSmSetNVAbsentCount: Set the non volatile Rx counter

| | |
|---|---|
| **Prototype** | void **ApplSmSetNVAbsentCount**( inmNmIndexType index, canunint8 value ) |
| **Parameter** | index Handle of the monitored receive message. Handle can be 0 to r-1 (r = number of all monitored receive messages) starting with 0.<br>value Value of counter |
| **Return code** | |
| **Functional description** | User specific function which is called if the state of reception will be changed (in Ap-plInmNmSetNVAbsentCount) and the non volatile storage of error counters is activated . |
| **Particularities and Limitations** | This function will only be called if configured. Please refer to the chapter about configu-ration. |

### 6.3.8 ApplSmSetNVMuteCount: Set the non volatile Tx counter

| | |
|---|---|
| **Prototype** | void **ApplSmSetNVMuteCount**( canunint8 value ) |
| **Parameter** | value Value of counter |
| **Return code** | |
| **Functional description** | User specific function which is called if the state of transmission will be changed (in ApplInmNmSetNVMuteCount) and the non volatile storage of error counters is activated . |
| **Particularities and Limitations** | This function will only be called if configured. Please refer to the chapter about configuration. |

### 6.3.9 ApplSmSetNVNerrCount: Set the non volatile Nerr counter

| | |
|---|---|
| **Prototype** | void **ApplSmSetNVNerrCount**( canunint8 value ) |
| **Parameter** | value Value of counter |
| **Return code** | |
| **Functional description** | User specific function which is called if the state of the Nerr supervision will be changed and the non volatile storage of error counters is activated . |
| **Particularities and Limitations** | This function will only be called if configured. Please refer to the chapter about configuration. |

### 6.3.10 ApplSmGetNVAbsentCount: Get the non volatile Rx counter

| | |
|---|---|
| **Prototype** | Canuint8 **ApplSmGetNVAbsentCount** ( inmNmIndexType index ) |
| **Parameter** | Index Handle of the monitored receive message. Handle can be 0 to r-1 (r = number of all monitored receive messages) starting with 0. |
| **Return code** | Value of non volatile Rx counter. |
| **Functional description** | User specific function which is called when the non volatile counters are initialised in ApplInmNmInitVolatileCounters( ) and is called in ApplInmNmGetNVAbsentCount to get the absent counter value. |
| **Particularities and Limitations** | This function will only be called if the storage of error counters is activated. Please refer to the chapter about configuration. |

### 6.3.11 ApplSmGetNVMuteCount: Get the non volatile Tx counter

| | |
|---|---|
| **Prototype** | Canuint8 **ApplSmGetNVMuteCount** ( void ) |
| **Parameter** | |
| **Return code** | Value of non volatile Tx counter. |
| **Functional description** | User specific function which is called when the non volatile counters are initialised in ApplInmNmInitVolatileCounters( ). |
| **Particularities and Limitations** | This function will only be called if the storage of error counters is activated. Please refer to the chapter about configuration. |

### 6.3.12 ApplSmGetNVNerrCount: Get the non volatile Nerr counter

| | |
|---|---|
| **Prototype** | Canuint8 **ApplSmGetNVNerrCount** ( void ) |
| **Parameter** | |
| **Return code** | Value of non volatile Nerr counter. |
| **Functional description** | User specific function which is called when the non volatile counters are initialised in ApplInmNmInitVolatileCounters( ). |
| **Particularities and Limitations** | This function will only be called if the storage of error counters is activated. Please refer to the chapter about configuration. |

### 6.3.13 ApplSmTrcvOn: Switch on the transceiver

| | |
|---|---|
| **Prototype** | Void **ApplSmTrcvOn** ( void ) |
| **Parameter** | |
| **Return code** | |
| **Functional description** | User specific function which is called when the Netmanagement is switched on. Calls the hardware specific routines to activate the transveiver. |
| **Particularities and Limitations** | |

### 6.3.14 ApplSmTrcvOff: Switch off the transceiver

| | |
|---|---|
| **Prototype** | Void **ApplSmTrcvOff** ( void ) |
| **Parameter** | |
| **Return code** | |
| **Functional description** | User specific function which is called when the Netmanagement is switched off. Calls the hardware specific routines to deactivate the transveiver. |
| **Particularities and Limitations** | |

### 6.3.15 ApplNwmBusOff: Bus off indication

| | |
|---|---|
| **Prototype** | Void **ApplNwmBusOff** ( void ) |
| **Parameter** | |
| **Return code** | |
| **Functional description** | User specific function which is called when a Bus Off situation is detected. |
| **Particularities and Limitations** | |

### 6.3.16 ApplNwmBusOffEnd: Bus off recovery ended

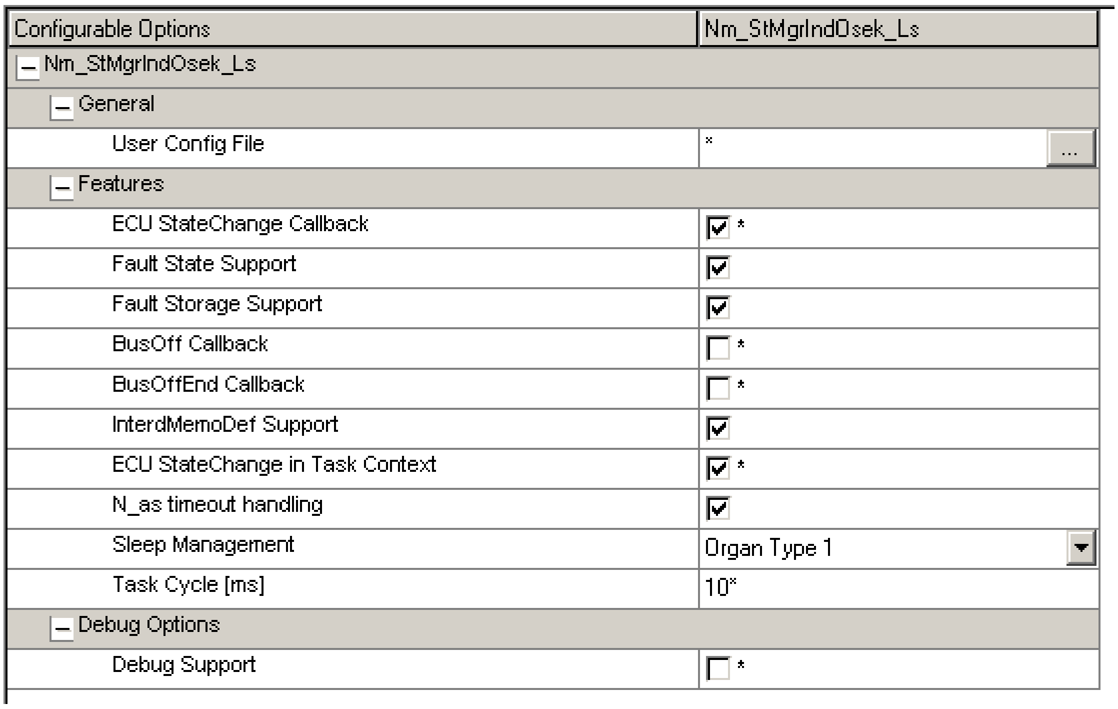
| | |
|---|---|
| **Prototype** | Void **ApplNwmBusOffEnd** ( void ) |
| **Parameter** | |
| **Return code** | |
| **Functional description** | User specific function which is called when the Bus Off recovery is finished. |
| **Particularities and Limitations** | |

### 6.3.17 ApplSmFatalError: Error in assertion occurred

| | |
|---|---|
| **Prototype** | Void **ApplSmFatalError** ( canuint8 errorcode) |
| **Parameter** | errorcode Error code for SM assertion (defined in header file) |
| **Return code** | |
| **Functional description** | The code contains self testing code sequences which check necessary preconditions of states, parameters, … during runtime. If an unexpected value is detected this function is called. |
| **Particularities and Limitations** | The function is only used if runtime checks (debug switch) are activated in the generation tool. |

# 7 Configuration of the station-manager

The complete configuration can be done by using the generation tool *CANGen*. The options are described below. The configuration data is generated into the file stat_cfg.h.



| Configurable Options | Nm_StMgrIndOsek_Ls |
|---|---|
| Nm_StMgrIndOsek_Ls | |
| General | |
| User Config File | * |
| Features | |
| ECU StateChange Callback | ✓ * |
| Fault State Support | ✓ |
| Fault Storage Support | ✓ |
| BusOff Callback | ☐ * |
| BusOffEnd Callback | ☐ * |
| InterdMemoDef Support | ✓ |
| ECU StateChange in Task Context | ✓ * |
| N_as timeout handling | ✓ |
| Sleep Management | Organ Type 1 |
| Task Cycle [ms] | 10* |
| Debug Options | |
| Debug Support | ☐ * |

The Station-manager configuration dialog

## 7.1 General Configuration

User Config File:

If in any case you need some special configuration, you can create a user specific configuration file and enter the path here. The Generation Tool  inserts the file automatically to the normal configuration.

## 7.2 Status Callback functions

The following callback functions can be activated:

### 7.2.1 ECU StateChange Callback:

Whenever the state of the ECU is changing (wakeup, sleep, normal,…) the application is notified by the ***ApplSmStatusIndication*** callback function.

### 7.2.2 Fault State Support:

This switch enables the callback functions

ApplSmStatusIndicationRx

ApplSmStatusIndicationTx

ApplSmStatusIndicationNerr

Whenever the state of a supervised ECU, the Mute state of the own ECU or the Nerr supervision state changes the corresponding callback function is called with the states OK, Failure, Confirmed Failure

### 7.2.3 Fault Storage Support:

Enables the non volatile storage support for network errors. If enabled, the station manager indicates to the application that for a supervised ECU, the Mute state or the Nerr state a non volatile error must be stored due to a confirmed failure detection. The following callbackfunctions are called in the application:

ApplSmSetXXXAbsentCount

ApplSmGetXXXAbsentCount

where XXX is either *Absent, Mute or Nerr*.

### 7.2.4 Bus Off Callback Support :

Enable or disable the notification of the application about the beginning of the Bus Off recovery sequence. If this switch is active, the function ***ApplNwmBusoff()*** is called by the Station Manager.

### 7.2.5 Bus Off End Callback Support:

Enable or disable the notification of the application about the end of the Bus Off recovery sequence. If this switch is active, the function. ***ApplNwmBusoffEnd()*** is called by the Station Manager.

### 7.2.6 Use Flag InterdMemoDef

Enables the handling of the InterdMemoDef flag. The application has to provide the function ***ApplSmGetInterdMemoDef* (…)** where it returns the current state of the signal IntedMemoDef.

### 7.2.7 ECU State Change in Task Context

This switch is enabled by default. With this setting the state change requested by the BSI ECU by the message Commande_BSI is performed on task level and the transition time thus dependent on the cycle time of the SmTask().

IF the switch is disabled, the transition is performed immediately when receiving the Commande_BSI message. If the system is runnging in interrupt mode, be aware that the state change callback ( if configured ) is executed in interrupt context.

### 7.2.8 N_as Timeout Handling

If enabled the N_as timeout handling is supported by the station manager.

This is only possible if the task cycle time is less or equal 5ms.

### 7.2.9 Sleep Management

Select the Organ type of your ECU. Please note that “No Sleep management” is not supported!

### 7.2.10 Task cycle

Adjust the task cycle time of SmTaski() in ms.

### 7.2.11 Debug Support

If enabled Station manager specific assertions are activated.

If activated, the callback function ApplSmFatalError must be provided by the Application.

# 8 Database attributes

Some supplementary data and configurations are stored in the database as attributes of specific objects. To provide a correct generation of configuration files, these attributes have to be set to specific values.

## 8.1 Receive Message Attribute

| | |
|---|---|
| **Name** | NmInmMaster |
| **Type** | Enum |
| **Elements** | No, Yes |
| **Value** | No, Yes |
| **Particularities** | Set this attribute to yes for the message containing the signal containing the ECU state request ( e. g. /RCSM/: Master_WakeUpSleepCmd ; [INM PSA]: Phases_de_vie ) |

| | |
|---|---|
| **Name** | NmMessage |
| **Type** | Enum |
| **Elements** | No, Yes |
| **Value** | No, Yes |
| **Particularities** | Set this attribute to yes for the messages where the Mode Degrade Fonctionnel has to be supervised ( s. chapter 3.5 ) |

| | |
|---|---|
| **Name** | GenMsgMinAcceptLength |
| **Type** | Integer |
| **Elements** | - |
| **Value** | 0--8 |
| **Particularities** | Set this value to Lmin of the specific message. If no Lmin value is given set it to DLC |

| | |
|---|---|
| **Name** | GenMsgTimeoutTime_<ECU> |
| **Type** | Integer |
| **Elements** | - |
| **Value** | 0—10000 |
| **Particularities** | If the message is supervised by your ECU ( in terms of NM ), set this attribute to the timeout time of the supervised message (in general it’s 3 times the cycle time).<br><br><ECU> is the name of the ECU you develop. |

| | |
|---|---|
| **Name** | GenMsgStartDelayTime |
| **Type** | Integer |
| **Elements** | - |
| **Value** | 0—10000 |
| **Particularities** | This attribute gives the delay time in ms before the message is transmitted the first time after enabling the IL TX path by function ILTxStart() |

## 8.2 Signal Attribute

| | |
|---|---|
| **Name** | GenSigSpecialFunction |
| **Type** | String |
| **Elements** | - |
| **Value** | NmOsekI_Version ( attention: letter after Osek is <u>not</u> minor **L** but major **i** ) |
| **Particularities** | This attribute has to be set for at least one signal of the version |

| | |
|---|---|
| **Name** | GenSigSpecialFunction |
| **Type** | String |
| **Elements** | - |
| **Value** | NmOsekI_Nas ( attention: letter after Osek is <u>not</u> minor **L** but major **i** ) |
| **Particularities** | This attribute has to be set for the signal containing the value of the N_as counter to be transmitted to the BSI ECU |

# 9 Precautions

This chapter describes specific precautions which have to be taken into account.

## 9.1 Calling CanSleep(…); within status callback

Some specific CAN drivers may run into a endless loop or a hardware trap when calling *CanSleep(..);* inside a CAN interrupt. The callback function *ApplSmStatusIndication(…);* is called in a CAN interrupt context if the feature <ECU state change in Task context> (ref.chapter 7.2.7) is deactivated.

Please make sure not to call *CanSleep(…);* inside this callback function if the state change is not in task context!