Protocol interface manual

CANopen Master

Hilscher Gesellschaft für Systemautomation mbH
Rheinstraße 15
D-65795 Hattersheim
Germany

Tel. +49 (6190) 99070
Fax. +49 (6190) 990750

Sales: +49 (6190) 99070
Hotline and Support: +49 (6190) 990799

Sales email: sales@hilscher.com
Hotline and Support email: hotline@hilscher.com

Homepage: http://www.hilscher.com

Index	Date	Version	Chapter	Revision
1	10.05.96	1.000	all	created
2	09.09.97	1.005	all	Extention of the 'intel_motorola' byte in bus parameters.Extention of the slave PDO size indicator 'bLength' in the node parameter data set . Changing the I/O offset interpretation from word to byte offset address.
3	27.11.97	1.005	all	correction of the CAN-HEADER specifc declarations
4	26.01.98	1.005	all	new error numbers in chapter 3.3, new Function Up_Download
5	30.11.98	1.023	all	- new chapter 'General procedure how to get the DEVICE operative without SyCon' - new chapter 'Deleting existing data base in the DEVICE' - new chapter 'Starting and Stopping Communication during Runtime' - new protocol parameter usWatchDogtime included
6	11.06.99	1.034		- new function CANM_Send_CANMsg_Transparent
7	28.02.00	1.041		- renaming CANM_ definitions to COM_ definition to fit in our standard nomenklatura - correction of Bus-Parameter download message - new chapter access to dbm-file via dbm32.dll - new message COM_Reset_Err_Event - new error counter in global bus status field named Rx_Overflow
8	13.08.00			- new chapter for sending and receiving 29Bit Identifier messages - new table BUS_CAN_EXT for dbm32dll - changing Type_of_Pdo definition in chapter Coding of the Node Parameter data set.
9	30.07.01	1.048		- new chapter Coding of the Extended Bus Parameter Data Set and loading via message through HOST program - new parameter in the node parameter data set to enable/disable parts of the boot up sequence between DEVICE and node during startup - new chaper access to dbm file of sycon by embedded HOST applications
10	06.09.01	1.049		- new chapter CAN layer 2 access
11	30.10.01	1.050		- Abort reason code coming from a remote station during SDO-transfer is now handed over to the HOST application - new NMT message added
12	11.01.02	1.060		- start/stop devices connection online.
13	26.08.02	1.060		- command COM_Upload appended
14	30.10.02			- feature List of Master in appendix added - new parameter Heartbeat added in extended bus parameter chapter 5.2.5.2 - new parameter in node paraeter data to select between nodeguarding and heartbeat.

Continued on next page.

Index	Date	Version	Chapter	Revision
15	30.09.02	1.064		- support of 29Bit for all EC1 products in chapter "29Bit capability"
16	09.07.03	1.070		- SYNC timer can be configured to value 0 also
17	05.02.04	1.070		- correction of task token 2 to 3 in message "receive CAN message" - correction of acceptence mask and code in chapter "extended bus parameter" from 0x1FFFFFFF to 0xF8FFFFFF - correction of the "extended bus parameter" download message

1 Indroduction

This manual describes the user interface of CANopen for the communication interfaces and the communication module. The aim of this manual is to support the integration of these devices into own applications based on device driver function or direct access into the dual-port memory.

The general mechanism for the data transfer is protocol independent and described in the 'general definitions' of the toolkit manual.

All parameters and data have basically the description LSB/MSB. This corresponds to the convention of the Microsoft-C-compiler.

1.1 Protocol Signification

To manage the CANopen protocol 2 tasks are involved in the system. Therefore following entries for the protocol signification in the variables `TaskiName` of the dual-port memory are done:

```
Task2Name: 'PLC     '
Task3Name: 'CAN     '
```

1.2 The Process Data Interface

The DEVICE handles up to 3584 bytes send and 3584 bytes receive process data in the lower 7 kbyte of the dual-port memory. To exchange the data between the DEVICE and the HOST use the device driver function `DevExchangeIO()`or read and write directly into these locations.

2 Protocol Parameter

Some important parameters can be handed over to the DEVICE online. They have a higher priority than the static parameters in the internal FLASH memory.

2.1 Using Device Driver Function to Write

To hand over these parameters use the device driver function `DevPutTaskParameter()`. For parameter `usNumber` use value 2, because the parameters must handed over to the task 2. For parameter `usSize` use value 3, to fix the length of the structure. Point the parameter `pvData` to the following structure below.

```
typedef struct COM_PLC_PARAMETERtag {
  unsigned char   bMode;
#define CANM_SET_MODE_BUFFERED_DEVICE_CONTROLLED 1
#define CANM_SET_MODE_UNCONTROLLED               2
#define CANM_SET_MODE_BUFFERED_HOST_CONTROLLED   3
  unsigned short  usWatchDogTime;
  unsigned char   abReservedA[5];
  unsigned char   abReservedB[8];
} COM_PLC_PARAMETER;
```

2.2 Direct Write Access in Dual-port Memory

First the parameters must be written down into the corresponding area of the dual-port memory. Then a warmstart command must be activated by setting the `Init` bit in the variable `DevFlags`. Then the DEVICE will set them valid. (See the chapter 'initialization of the DEVICE' in the toolkit manual 'general definitions' for handle of the init procedure).

structure element	type	address	parameter
bMode	byte	1EC0H	process data delivery (0,1,2,3,4)
usWatchDogTime	word	1EC1H	HOST-supervision time in multiples of a msec.

Protocol parameters in area `Task2Parameter`

2.3 Explanation of the Protocol Parameters

The first parameter fixes the handshake mode of the process data.. Valid entries are 1,2,3. See the chapter 'Handshake mode of process data delivery' in the toolkit manual 'general definitions', for the explanation of the different modes.

The parameter `usWatchDogTime` fixes the time in multiples of 1msec. the DEVICE has to supervise the HOST program if it has started the HOST-watchdog functionality once. Read in manual 'Toolkit General definitions' how to activate and deactivate the DEVICE and HOST supervision.

3 Controlled Start and Stop of I/O Connection

The slave connection status can be changed by the HOST online if the DEVICE is in state OPERATE.
The table that can be written to consists of an array of 128 bit, each for one node, to control the current I/O communication of the node. The bit field is evaluated by the DEVICE at least every millisecond. If the corresponding bit is cleared = logial '0', the DEVICE will stop the communication to the node. A logical '1' will start the communication and the DEVICE will instantaneously tries to reestablish the communication.

3.1 Using Device Driver Function to Write

To hand over these status information use the device driver function `DevReadWriteDPMRaw()`. For parameter `usMode` use value 2 = `PARAMETER_WRITE` to select the write option. For parameter `usOffset` use value `2F0H`, since this parameter selects the offset within the last kilobyte of the dual-port memory. The length indicator `usSize` which represents the size of the structure must be set to value 16. Point the parameter `pvData` to the following structure below.

```
typedef struct COM_NODE_CONN_STATUStag {
  unsigned char   abConnStatus[16];
} COM_NODE_CONN_STATUS;
```

3.2 Direct Write Access into the Dual-Port Memory

Write the parameter to the offset below.

structure element	type	address	parameter
abConnStatus	bit array	1EF0H-1EFFH	connection status of the devices

Online Status parameter in area `Task2Parameter`

3.3 Explanation of the Status parameter

Bit / Offset	D7	D6	D5	D4	D3	D2	D1	D0
1EF0H	7	6	5	4	3	2	1	
1EF1H	15	14	13	12	11	10	9	8
1EF2H	23	22	21	20	19	18	17	16
...								
1EFFH	127	126	125	124	123	122	121	120

Relation table between device node address and the Connection Status bit

4 Protocol States

The protocol states built by the DEVICE form the diagnostic interface between the HOST and the CANopen.
The established structure informs about global busstates as well as individual states of the managed nodes. To hold the information preferably compact, the node specific informtion are held in state bitfields.
The first 4 state variables in the structure informs about global bus state informations. They are followed by a bus error event and and a bus off counter. After this field an unused reserved area of 8 byte is following. The next 16 bytes characterize the node as configured and the next 16 bytes characterize every slave station as active or inactive, followed by 16 bytes which serve to refer the diagnostic bit of every node station.

4.1 Using Device Driver Funtions

Use the device driver function `DevGetTaskState()` to read the states. For parameter `usNumber` use value 2, because the parameter must be read from the task state area of the task 2. For parameter `usSize` use value 64, which is the length of the structure below. Point `pvData` to the following structure.

```
typedef struct COM_DIAGNOSTICStag {
  struct
  {
    unsigned char bCtrl     : 1;
    unsigned char bAClr     : 1;
    unsigned char bNonExch  : 1;
    unsigned char bFatal    : 1;
    unsigned char bEvent    : 1;
    unsigned char bNRdy     : 1;
    unsigned char bTOut     : 1;
    unisgned char bReserved : 1;
  } bGlobalBits;

  unsigned char   bCAN_State;

  struct
  {
    unsigned char bErr_rem_adr;
    unsigned char bErr_event;
  } tError;

  unsigned short  usBus_Error_Cnt;
  unsigned short  usBus_Off_Cnt;
  unsigned short  usMsg_Time_Out;
  unsigned short  usRx_Overflow;
  unsigned char   abReserved[4];

  unsigned char   abNd_cfg[16];
  unsigned char   abNd_state[16];
  unsigned char   abNd_diag[16];
} COM_DIAGNOSTICS;
```

4.2 Direct Read Access in Dual-port Memory

Read the bus state structure directly from the following dual-port memory location:

variable	type	address	short signification
global_bits	1 byte	1F40H	global error bits
CAN_state	1 byte	1F41H	main state of the master system
Err_Node_adr	1 byte	1F42H	faulty node address
Err_event	1 byte	1F43H	error specification
Bus_Error_Cnt	word	1F44H	number of detected bus error limit oversteps
Bus_Off_Cnt	word	1F46H	number of CAN-chip reinitializations
Msg_Time_Out	word	1F48H	number of canceled CAN messages, because of getting no message acknowledging partner
Rx_Overflow	word	1F4AH	number of indicated Receive Message Overflow coming from the CAN chip
reserved	4 bytes	1F4CH-1F4FH	reserved error variables
Nd_cfg	16 bytes	1F50H-1F5FH	see the table below
Nd_state	16 bytes	1F60H-1F6FH	see the table below
Nd_diag	16 Bytes	1F70H-1F7FH	see the table below

Global bus state field in area `Task2State`

4.3 Explanation of the Protocol States

- global_bits

D7	D6	D5	D4	D3	D2	D1	D0
0	TOUT	NRDY	EVE	FAT	NEXC	ACLR	CTRL

CONTROL-ERROR:
parameterization error

AUTO-CLEAR-ERROR:
DEVICE stopped the communication to all nodes and reached the auto-clear end state

NON-EXCHANGE-ERROR
At least one node has not reached the data exchange state and no process data are exchange with it.

FATAL-ERROR:
Because of heavy internal error, no further bus communication is possible

EVENT-ERROR:
The DEVICE has detected transmission errors. The number of detected events are fixed in the Bus_Error_Cnt and Bus_Off_Cnt variables. The bit will be set when the first event was detected and will not be deleted any more.

HOST-NOT-READY-NOTIFICATION:
Indicates if the HOST program has set its state to operative or not. If the bit is set the HOST program ist not ready to communicate

TIMEOUT-ERROR:
The DEVICE has detected an overstepped timout supervision time of at least one CAN message to be sent. The transmission of this message was aborted. The data is lost. It's an indication that no other CAN device was connected or responsive at this time to acknowledge the sent message requests. The number of detected timeouts are fixed in the Msg_Time_Out variable.The bit will be set when the first timeout was detected and will not be deleted any more.

reserved

The bit field serves as collective display of global notifications. Notified errors can either occur at the DEVICE itself or at the nodes. To distinguish the different errors the variable `err_node_adr` contains the error location (address), while the variable `err_event` is fixing the corresponding error number. If more than one error is determined, the error location will always show the lowest faulty bus address.

- Variable CAN_state

This variable represents the main state of the master system. Following values are possible:

00H: state OFFLINE
40H: state STOP
80H: state CLEAR
C0H: state OPERATE

- Variable Err_Node_adr

If either the bits `Ctrl, Aclr` or `NData` are set, this variable fixes the nearer location of the error. If the source of the error is determined inside the DEVICE itself, the value 255 is written in. Else the faulty node address is written in directly.

- Variable Err_Event

This variable is a closer specification of the error of the faulty participant reported in `Err_Node_adr` . See the table below for the nearer explanation of the error numbers.

- Variable Bus_Error_Cnt

This variable is incremented if the error counter of the CAN chip has reached the mircocontroller warning limit because of bus errors.

- Variable Bus_Off_Cnt

This variable is incremented if the CAN chip reports that he is no longer involved in bus activities because of overstepped internal bus error counters and must be reinitialised.

- Variable Msg_Time_out

Each CAN message is supervised by the card to be sent during 20ms by the CAN chip. If not possible, because the chip for example gets no acknowledging partner on the bus, the counter is incremeted by one.

- Variable Rx_Overflow

If the firmware itself is not fast enough for receiving all CAN message coming via the network in time, this overflow counter is incremented per lost message

- Variable reserved

This data block is reserved.

- Variable Nd_cfg

This variable is a field of 16 bytes and contains the paremeterization state of each node station. The following table shows, which bit is related to which node station address:

Bit / Offset	D7	D6	D5	D4	D3	D2	D1	D0
750H	7	6	5	4	3	2	1	0
751H	15	14	13	12	11	10	9	8
752H	23	22	21	20	19	18	17	16
...								
75FH	127	126	125	124	123	122	121	120

Table of the relation between node address and the Nd_Cfg bit

If the Nd_cfg bit of the corresponding node is logical

'1', the node is configured in the master, and serviced in its states.

'0', the node is not configured in the master

- Variable Nd_state

This variable is a field of 16 bytes and contains the state of each node station. The following table shows, which bit is related to which node address:

Bit / Offset	D7	D6	D5	D4	D3	D2	D1	D0
760H	7	6	5	4	3	2	1	0
761H	15	14	13	12	11	10	9	8
762H	23	22	21	20	19	18	17	16
...								
76FH	127	126	125	124	123	122	121	120

Table of the relation between node address and the Nd_state bit

If the Nd_state bit of the corresponding node is logical

'1', node is operating, node guarding reports no error

'0', node is not operating, because he is not configured or has an error

The values in variable Nd_state are only valid, if the master runs the main state OPERATE.

- Variable Nd_diag

This variable is a field of 16 bytes containg the diagnostic bit of each node. The following table shows the relationship between the node address and the corresponding bit in the variable Nd_diag.

Bit / Offset	D7	D6	D5	D4	D3	D2	D1	D0
770H	7	6	5	4	3	2	1	0
771H	15	14	13	12	11	10	9	8
772H	23	22	21	20	19	18	17	16
...								
77FH	127	126	125	124	123	122	121	120

Table of the relationship between node address and the Nd_diag bit

If the Nd_diag bit of the corresponding node is logical

'1', newly received emergency message are available in the internal diagnostic buffer or one of the diagnostics bit of the node has changed. This data can be read out by the HOST with a message which is described in the chapter 'The message interface' in this manual.

'0', since the last diagnostic buffer read access of the HOST, no values were change in this buffer.

The values in variable Nd_diag are only valid, if the master runs the main state OPERATE.

	Nd_state = 0	Nd_state = 1
Nd_diag = 0	- node not operation, no DataIOExchange between master and node - Perhaps this slave is not configured	- node is present on the bus, node guarding active - PDO exchange between master and node happens as configured
Nd_diag = 1	- node is not operating, node guarding failed - The master holds newly received diagnostic data in the internal diagnostic buffer	- node is present on the bus, node guarging is active, PDO exchange - The master holds newly received diagnostic data in the internal diagnostic buffer

Relationchip between Nd_state bit and Nd_diag bit

The following error numbers are valid for `Err_event`, if `Err_Rem_Adr` is 255:

err_event	signification	error source	help
0	no error		
52	unknown handshake mode was configured	initialization	see chapter 'Protocol parameters' for valid entries
56	baudrate out of range	project planning	contact technical support
60	double node address was configured	project planning	contact technical support
220	HOST-watchdog error	HOST program	check user program or watchdog time
210	no database	project planning	execute download of the data base
212	faulty reading of a data base	device	execute download of data base again

The following error numbers are valid for `Err_Event`, if `Err_Rem_Adr` is unequal 255:

err_event	signification	error source	help
30	guarding failed	node	check if node is connected
31	node has change his state and is not longer operational	node	reset node
32	sequence error in guarding protocol	node	reset node
33	no response to a configured remote frame PDO	node	check if node is able to handle remote frames
34	no response of the node during its configuration	node	check if node is connected and operational
35	the master configured node profile number is different from the node profile number	project planning	check the supported profile number of the node:I/O,encoder,etc.
36	the master configured node device type is different from the node device type	project planning	check the supported services of the node.
37	unknown SDO response received	node	node not compatible to CiA protocol specification
38	length indicator of received SDO message unequal 8	node	node not compatible to CiA protocol specification
39	node not handled, node in stop	device	activated Auto_Clear mode or host not ready

5 The Message Interface

The following send and receive messages are exchanged with the DEVICE via its mailboxes in the structure like it is described in the chapter 'definition of the messageinterface' in the toolkit manual.

To put and get messages to respectively from the DEVICE through its mailboxes use the device driver functions `DevPutMessage()` or `DevGetMessage()`. With direct access to the dual-port memory you must write the message in the `DevMailbox()` or read the message out of the `HostMailbox()` with the mechanism described in the toolkit manual.

The structures of this messages and its values are described in the sections below.

5.1 The PLC-Task

The PLC-Task manages the process input and output data and handle the steering of the CAN cycles corresponding the parameterization. Therefore the task communicates with the CAN-Task and starts the nodes handling according to the parametrized operation mode. The task has implemented the following functions:

- activates CAN cycles.

The task manages following message command:

Shared memory	write consistent data block into the send process data `SndPd` or read consistent data block from `RecvPd`.

5.1.1 Shared Memory

command message			
variable	type	value	signification
msg.rx	byte	2	receiver = PLC-TASK
msg.tx	byte	16	transmitter = user at HOST
msg.ln	byte	 8 8+m	length of the message read access write access
msg.nr	byte	j	number of message (optional)
msg.a	byte	0	no answer
msg.f	byte	0	no error
msg.b	byte	17	command = COM_Shared_Memory
msg.e	byte	0	unused
msg.device_adr	byte	0	device address unused
msg.data_area	byte	 0 1 2	data area: msg.function decides the data area receive process data area send process data area
msg.data_adr	word	 0-1791 0-3583 0-1791	address offset refer to the data type word-offset address Byte-offset address word-offset address if bit access
msg.data_idx	byte	0-15	bitposition within the word offset address if bit access
msg.data_cnt	byte	m	count of read or write data referring to the data type
msg.data_type	byte	 6 5,10 14	data type: TASK_TDT_UINT16: word TASK_TDT_UINT8: octet-string TASK_TDT_BIT: bit
msg.function	byte	 1 2	function: TASK_TFC_READ = read access TASK_TFC_WRITE = write access
msg.d[0]	byte	x	write access: first data to be written read access: unused
...	...	...	...
msg.d[m-1]	byte	z	write access: last data to be written read access: unused

The command serves the user program to write data of a definite length into the send process data buffer or to read from the receive process data. The command can be used in all operation modes.
With the command the data types word, bytes or bits can be selected. The firmware of the DEVICE guarantees that the read or write access of the data will be done safely during two active PDO transfers.

This command ia an other oportunity to get access to the datas in the dual-port memory. It is slower as the direct reading or writing the dual-port memory. The advantage is, that you can use the same mechnism as in a message oriented protocol. We use this message in our diagnostic tools.

The data type is fixed in the byte `msg.data_type`. Only the values decimal 6 for words, 5 or 10 for byte strings and 14 for bits are allowed.

The read access is distinguished from the write access in the byte `msg.function`. A 1 is valid for read access and 2 for write access.

The data area to be read from, is fixed in `msg.data_area`. 1 is valid for the receive process data buffer and 2 for the send process data buffer. In case of the value 0, the value placed in `msg.function` decides the data area automatically.

The count of the data to be read or to be written is fixed by the value of `msg.data_cnt`. The count refers to the chosen data type. Maximum permitted values are 119 for words, 239 for byte and 255 for bits.

The offset address is fixed in the word `msg.data_adr`. The specified address must be refered on the chosen data type and is interpreted from the DEVICE as the relative address to the start address in the send process data or the receive process data. The maximum values are decimal 1791 for word, 3583 for byte and 1791 for bit access. In case of bit access the value in `msg.data_idx` additionally fixes the relative offset in the word to be read or to be written. For the other accesses the value doesn't have any meaning.

The data at `msg.d[]` are unused if read access is chosen, while in case of a write access the send data must be written into it. Words must be written in Intel format - LSB before MSB - and bits must be put in there in packed form. For example to write 5 bits, the first databyte `msg.d[0]` must be placed in the bits 0-4 to be valid.

answer message to the user			
variable	type	value	signification
msg.rx	byte	16	receiver = user at HOST
msg.tx	byte	2	transmitter = PLC-Task
msg.ln	byte	 8+m 8 0	length of the message read access write access error
msg.nr	byte	j	number of message
msg.a	byte	17	answer = COM_Shared_Memory
msg.f	byte	0 f	no error error nummber see following table
msg.b	byte	0	no command
msg.e	byte	0	unused
msg.device_adr	byte	0	device address unused
msg.data_area	byte	 0 1 2	data area: datafnc decides the data area receive process data area send process data area
msg.data_adr	word	 0-1791 0-3583 0-1791	address offset refer to the data type word-offset address Byte-offset address word-offset address if bit access
msg.data_idx	byte	0-15	bit position within the word offset address if bit access
msg.data_cnt	byte	m	count of read or write data referring to the data type
msg.data_type	byte	 6 5,10 14	data type: TASK_TDT_UINT16: word TASK_TDT_UINT8: octet-string TASK_TDT_BIT: bit
msg.function	byte	 1 2	function: TASK_TFC_READ = read access TASK_TFC_WRITE = write access
msg.d[0]	byte	x	read access: first data be read write access: unused
...	...	...	...
msg.d[m-1]		z	read access: last data be read write access: unused

In the answer message the `msg.f` byte reports back the information, if the command could be executed. If the error byte is 0, then the service could be finished without any error.

The extended telegram header of the answer message is not be changed if the command could be executed correctly, so that the user can assign the answer message unequivocally to the sent command message.

The convention for the data `msg.d[]` coming back in the answer message is the same one as the convention for the command messages. That means that words will be placed in Intel format and bits are placed in packed format.

The byte `msg.f` in the answer message can have the following values:

error number msg.f	signification
0	no error, command executed
162	illegal address area
163	data address together with the data count results an overflow in the buffer length
165	illegal data count
166	unknown data type
167	unknown function

5.2 The CAN-Task

The CAN-Task includes the two functional blocks:

- node handler
- CAN message queue management
- mapping of the physical addresses of the data to the logical addresses of the data in the dual-port memory.

For every node station one state machine is implemented. These indiviual state machines named node handler are managing the several states of each node like getting diagnostic, doing parameterization, doing configuration and doing the process data exchange.

The CAN-Task has implemented the following messages for the HOST:

`COM_Start_Seq`	start download of parameters
`COM_End_Seq`	end of download
`COM_Download`	download of parameters
`COM_Node_Diag`	read internal saved diagnostic datas the one slaves

The CAN-Task needs the configuration data for the CANOpen.

Normally the configuration will be downloaded by the SyConCO configuration tool statically into the FLASH memory. The task reads out this data block when the DEVICE starts up. If all parameters are valid the task starts its node handlers and goes into the mode OPERATE.

If no static download of the configuration data is wished, all these data can be handed over online to the DEVICE by a message download from the HOST program. But before doing this, you have to prevent the DEVICE to start up with possible downloaded static parameters. This can be done by deleting the data base 'CANopen' with the ComPro tool before. Then the DEVICE starts up without finding any configuration data file and the online message download can be done by the HOST program like it is described in the following chapters.
NOTE! If no data base exists on the DEVICE, the DEVICE must be initialized with protocol parameters (see chapter: protocol parameters) before the message download is done, to fix the process data handshake mode.

ANNEX: We recommend to get the CANopen draft specification named CiA Draft Standrad 301, when the online download of the parameters is used. The most configuration data containments of the messages have exactly the same meaning and functionality like it is described in the norm specification.

5.2.1 Starting and Stopping Communication during Runtime

5.2.1.1 Using Device Driver Function to Write

Use the function `DevSetHostState()` together with the parameter `HOST_NOT_READY` to stop the network communication. Use the parameter `HOST_READY` to start or restart the communication.

5.2.1.2 Direct Write Access in Dual-Port

To start and stop the communication of the DEVICE you have to clear and set the bit `NotRdy` in the cell `bDevFlags`. Clearing the bit will start the network communication while setting the bit will stop it.

ATTENTION: Stopping the communication will always cause a reset of the network modules output data.

5.2.2 Deleting Existing Data Base in the DEVICE

Normally the configuration will be downloaded by the SyCon configuration tool statically into the FLASH memory. The DEVICE reads out this data block during its startup. If all parameters are valid the DEVICE starts its slave handlers and goes into the mode OPERATE. Then the message download procedure like it is described in the chapters below can not be used any more.

If no static download of the configuration data is wished, all these data can be handed over online to the DEVICE by a message download from the HOST program using the functions Start,End and Download. But before doing this, you have to prevent the DEVICE to start up with possible downloaded static parameters. This can be done by deleting the data base by message service before. Then the DEVICE starts up without finding any configuration data base and then the online message download can be proceeded by the HOST program like it is described in the following chapters.
IMPORTANT NOTE! If no data base exists within the DEVICE, the DEVICE must be initialized with protocol parameters (see chapter: protocol parameters) before the message download is done, to fix the process data handshake mode and the storage format etc.work modules output data.

	command message			
	variable	type	value	signification
Message header	msg.rx	byte	0	receiver = RCS-Task
	msg.tx	byte	16	transmitter = user at HOST
	msg.ln	byte	2	length of the message
	msg.nr	byte	j	number of the message
	msg.a	byte	0	no answer
	msg.f	byte	0	no error
	msg.b	byte	6	command = data base access
	msg.e	byte	0	extention, not used
Service header	msg.d[0]	Byte	4	mode = delete data base
	msg.d[1]	Byte	8	startsegment of the data base

	answer message			
	variable	type	value	signification
Message header	msg.rx	byte	16	receiver = user at HOST
	msg.tx	byte	0	transmitter = RCS-Task
	msg.ln	byte	1	length of message
	msg.nr	byte	j	number of the message
	msg.a	byte	6	answer = data base access
	msg.f	byte	f	error, state
	msg.b	byte	0	no command
	msg.e	byte	0	extension, not used

The time for deleting the data base depents on the used FLASH memory, so sending back the answer message can take up to 3 seconds

5.2.3 COM_Start_Seq

	command message			
	variable	type	value	signification
Message header	msg.rx	byte	3	receiver = CAN-Task
	msg.tx	byte	16	transmitter = user at HOST
	msg.ln	byte	4	length of the message
	msg.nr	byte	j	number of the message
	msg.a	byte	0	no answer
	msg.f	byte	0	no error
	msg.b	byte	67	command = COM_Start_Seq
	msg.e	byte	0	extension, not used
COM_START_SEQ_REQUEST	msg.d[0]	Byte	0	Req_Adr, unused
	msg.d[1]	Byte	1-126	Area_Code, node address
	msg.d[2]	Word	0-65535	Timeout, mutiple of 1 ms

The command starts a blocked download in the stated `Area_Code`, if the node parameter file to download is larger than one message. To complete the download the command `CAN_End_Seq` must be called after finishing the download sequence.

	answermessage			
	variable	type	value	signification
Message header	msg.rx	byte	16	receiver = user at HOST
	msg.tx	byte	3	transmitter = CAN-Task
	msg.ln	byte	1	length of message
	msg.nr	byte	j	number of the message
	msg.a	byte	67	answer = COM_Start_Seq
	msg.f	byte	f	error, state
	msg.b	byte	0	no command
	msg.e	byte	0	extension, not used
COM_START_SEQ_CONFIRM	msg.d[0]	byte	240	Max_Len_Data_Unit

The value `Max_Len_Data_Unit` fixes the maximum length of the parameter `Data` for each following `CAN_Download` message.

Possible values for `msg.f` are the following:

error number msg.f	signification
0	no error
52	Area_Code unknown

See below the corresponding structures in the header file:

```
COM_START_SEQ_REQUEST
COM_START_SEQ_CONFIRM
```

5.2.4 COM_End_Seq

	command message			
	variable	type	value	signification
Message header	msg.rx	byte	3	receiver = CAN-Task
	msg.tx	byte	16	transmitter = user at HOST
	msg.ln	byte	1	length of the message
	msg.nr	byte	j	number of the message
	msg.a	byte	0	no answer
	msg.f	byte	0	no error
	msg.b	byte	69	command = COM_End_Seq
	msg.e	byte	0	extension, not used
COM_END_SEQ_REQUEST	msg.d[0]	byte	0	Req_Adr, unused

The command ends the blocked download and activates the data and the data check.

	answer message			
	variable	type	value	signification
Message header	msg.rx	byte	16	receiver = user at HOST
	msg.tx	byte	3	transmitter = CAN-Task
	msg.ln	byte	0	length of message
	msg.nr	byte	j	number of the message
	msg.a	byte	69	answer = COM_End_Seq
	msg.f	byte	f	error, state
	msg.b	byte	0	no command
	msg.e	byte	0	extension, not used

Possible values for `msg.f` are the following :

error number msg.f	signification
0	no error
59	data incomplete or faulty

See below the corresponding structure in the header file:

```
COM_END_SEQ_REQUEST
```

5.2.5 COM_Download

	command message			
	variable	type	value	signification
Message header	msg.rx	byte	3	receiver = CAN-Task
	msg.tx	byte	16	transmitter = user at HOST
	msg.ln	byte	m + 4	length of the message
	msg.nr	byte	j	number of the message
	msg.a	byte	0	no answer
	msg.f	byte	0	no error
	msg.b	byte	68	command = COM_Download
	msg.e	byte	0	extension, not used
COM_DOWNLOAD_REQUEST	msg.d[0]	byte	0	Req_Adr, unused
	msg.d[1]	byte	 1-126 127 128	Area_Code, node address master bus parameter extended master bus parameter
	msg.d[2]	word	0-760	Add_Offset, if sequenced node parameter download is used
	msg.d[4-240]	m bytes	0-255	Data[240] per message

This command allows to hand over the master bus parameters or the node parameter data files. This is commendable, if no static data base exists on the DEVICE and the parameterization should happen from the HOST program without using SyConCAN tool.

Two ways to download data files have been implemented. A data file can be downloaded either in one call (single download) or if it is to large in block calls (sequenced download) into an internal download area (length 1000 bytes). After the download cycle is finished completly the specified data is checked and copied afterwards into the task access area. Then the next download can be started into the freed download area.

The parameter `Area_Code` fixes the destination area (master parameter or slave parameter file). The offset in the download area where the data will be copied from the message is fixed in the variable `Add_Offset`.
If a node data file shall be transferred sequenced, the command `CAN_Start_Seq` must be activated before to initialize the download sequence. The sequence will be finished after the command `CAN_End_Seq` is called. Even then the parameters will be checked and be set valid if no error is recognized. The download of the bus parameters needs no sequenced download.

See below the corresponding structure in the header file:

```
COM_DOWNLOAD_REQUEST
```

	answer message			
	variable	type	value	signification
Message header	msg.rx	byte	16	receiver = user at HOST
	msg.tx	byte	3	transmitter =CAN-Task
	msg.ln	byte	0	length of message
	msg.nr	byte	j	number of the message
	msg.a	byte	68	answer = COM_Download
	msg.f	byte	f	error, state
	msg.b	byte	0	no command
	msg.e	byte	0	extension, not used

Possible values for `msg.f` are the following:

error number msg.f	signification
0	no error
48	timeout
52	area_code unknown
53	overstep of the buffer length
55	faulty parameter detected
57	sequence error
59	data incomplete or faulty
62	data check: node parameter size is too large
60	data check: double node address parameterized
64	data check: PDO data size is to olarge
61	data check: Add-Tab size is too large
63	data check: unknown PDO transmisstion type
65	data check: unknown baud rate
66	data check:COB-ID for SYNC messages out of range
67	data check:SYNC message timer reload value out of range
68	data check: input offset plus length too large
69	data check: output offset plus length too large
70	data check:PDO-cfg inconsistent to ADD-tab table
71	data check: length of ADD-Tab inconsistent
72	data check: length of whole data set inconsistent
73	data check: COB-ID for EMCY message out of range
74	data check: COB-ID for ND_GUARD message out of range
75	data check: Length indicator of one PDO out of range
76	data check: SDO data size is too large

5.2.5.1 Download of the Node Parameter Data Sets

5.2.5.1.1 Coding of the Node Parameter Data Set

Header Structure	variable name	type	explanation
COM_ND_PRM_HEADER	usNodeParaLen	word	length of whole data set inclusive the length parameter
	bNd_Flag	byte	bit field, to activate the parameter data set
	usAddTypeInfo	word	variable compared while the startup with the value in the node device type object 1000H
	usDevProfNum	word	variable compared while the startup with the value in the node device type object 1000H
	usCobIDEmcy	word	CAN identifier for nodes EMCY-message, to be configured while the start up process
	usCobIDGuard	word	CAN identifier for ND-GUARD message, to be configured while the start up process
	usGuardTime	word	guard time to supervise the node in multiples of 1msec
	bLifeTimeFactor	byte	multiplicator of the guard time
	bitConfigNdState	byte	bit array to enable and disable parts of node boot up sequence
	abReserved	1 byte	reserved
COM_ND_SDO_CFG_DATA	usSdoCfgDataLen	word	Length of the following SDO data inclusive the length of the size indicator
	atSdoCfgData	structure	see corresponding HEADER file for structure
COM_ND_PDO_CFG_DATA	usPdoCfgDataLen	word	Length of the following PDO data inclusive the length of the size indicator
	atPdoCfgData	structure	see corresponding HEADER file for structure
COM_ND_PRM_ADD_TAB	usAddTabLen	word	Length of the following add. tab data inclusive the length of the size indicator
	bInputCount	byte	number of following input offset addresses
	bOutputCount	byte	number of following output offset addresses
	ausIOOffsets[...]	word array	IO offset addresse of PDO data in IO area or DEVICE

The first length indicator `usNodeParaLen` fixes the length of the whole data block inclusive the length of the size indicator.

This variable is followed by a special bit field called `bNdFlag`. The topmost bit D7 = ACTIVE is declaring the parameter data set as active or inactive. If declared as not active, the DEVICE will not start any communication to this Node, but will reserve the process data for this Node if any is configured.
The next bit D6 = HRTBT decides if the DEVICE shall take either the traditional Guard Time Protocol or the Heartbeat Protocol to supervise a Node existance during PDO comminucation. If not set the Guarding Protocol is used , if set the Heartbeat Protocol is used.
The other bits in this byte are reserved for future use.

D7	D6	D5	D4	D3	D2	D1	D0
ACTIVE	HRTBT						

0 = for Node error control protocol, node guarding is used
1 = for Node error control protocol, Heartbeat is used

0 = node inactive in the actual configuration
1 = node active in the actual configuration

The next two words represent the contents 'device type object 1000H' of the node station. Both values `usAddTypeInfo` and `usDevProfNum` will be compared in the startup process of the node by reading the real object 1000H with SDO commands. If one of these two variables is not equal to the corresponding entry of the physically existing node, a runtime error will be generated and the node will not be brought into the OPERATIONAL state.

The next two CAN identifier values `usCobIDEmcy` and `usCobIDGuard` in the range of 0 to 2047dec are written into the node object directory during its startup. The Emergency COB ID is used later by the node to send Emergency Messages to the master like for example a low power indication. The standard calculation formula for the `usCobIDEmcy` is 80H + Node-ID. The Guarding COB-ID is the ID used later by master after the node is OPERATIONAL to supervise the node if it's still present in the network. The standard calculation formula for the `usCobIDGuard` is 700H + Node-ID.

The next two variables have different meanings based on the configured D6 = HRTBT bit in the variable `bNdFlag`.

HRTBT = 0:
The variable `usGuardTime` defines the time the DEVICE will poll the node at regular time intervals. The value is a multiple of 1msec. If the value for example is set to 500, the master will request the node at every half second. The `usGuardTime` is automatically configured by the master during the nodes startup process by writing this value into its object 100CH via SDO commands. If the value is 0 the guarding protocol is disabled.
The node life time is given by the `usGuardTime` multplied with the value `bLifeTimeFactor`. If the node has not been polled by the master during its life time, a remote node error is indicated by the node. So the `bLifeTimeFactor` can be seen as a safety factor the master gets time to send the node guarding request. The `bLifeTimeFactor` is automatically configured by the master during the nodes startup process by writing this value into its object 100DH via SDO commands. If the value is 0 the Life Guarding is disabled.

HRTBT = 1:
The variable `usGuardTime` defines the time the DEVICE will wait as Consumer for Heartbeat Indications of this Node.The value itself reflects the time in multiples of a millisecond. If the value for example is set to 500, the DEVICE will wait in maximum 500msecs for a Heatbeat Indication from this node till it would declare the Node as not present and faulty. If the value is configured to 0 the DEVICE will not supervise the Node and Heartbeat supervision as consumer and node existance check is disabled.
The value `bLifeTimeFactor` has no meaning in Heartbeat Protocol and is not used.

The bit field in byte `bitConfigNdState` configures the startup sequence the DEVICE is executing with the node. If the bit is logical '0' the corresponding service(s) is(are) active, if it is logical '1' it is (they are) deactivated.

D7	D6	D5	D4	D3	D2	D1	D0
TXRXPDO	START	SDO	EMCY	SYNC	GUARD	DEVICE	RESET
							Send Reset Node
						Check Device Profile	
					Configure Guarding Parameter		
				Configure Sync COB-ID			
			Configure Emergency COB-ID				
		Download of further Object data via SDO					
	Send Start Node Request						
Request first TX or send first RX PDO after initialization sequence is finished							

The SDO list serves to change data in the object directory of the node via SDO transfer. All object values be configured in this list are written into the located object diretory entry one after the other while start up the node. Each entry in the list must have the following structure:

Header Structure	variable name	type	explanation
COM_SDO_CFG_DATA	usIndex	word	index of the object to be written
	bSubindex	byte	subindex of the object to be written
	bLength	word	length of data to be written 1-4
	abSdoValue[...]	octect string	Sdo data[4]

The length of the SDO list fixes always the length of the SDO data field in bytes inclusive the size of the length indicator. If no objects should be changed, so that the SDO list is empty, a value of 2 results for the `usSdoCfgDataLen`.
If for example only the value in the object 6200H subindex 3 should be changed, the SDO entry should have the following data entries:

variable name	value
usSdoCfgDataLen	11
usIndex	0x6200
bSubindex	0x5
bLength	0x04
abSdoValue	0x10010108

The PDO list configures the CANopen process data messages (PDO) of this node. In the view of the master it enables the configured CAN identifier to be sent or to be received. So one entry in this list can configure either a transmit process data message = inputs in the view of the master or a receive process data message = outputs in the view of the master. See the following structure for one PDO entry:

Header Structure	variable name	type	explanation
COM_PDO_DATA	bTypeOfPdo	byte	defines the type of PDO
	bLength	byte	length of the PDO data
	usPdoCOBId	word	used COB-ID for this PDO
	bTransType	byte	type of transmission corresponding CiA drafr Standard 301
	usInhibitCycle	word	time the next PDO can be transmitted

The first value defines, if the configured PDO is a transmit or a receive PDO.

`bTypeOfPdo` = 0 :RXPDO outputs in the view of the master
`bTypeOfPdo` <> 0 :TXPDO inputs in the view of the master

One CANopen PDO message can hold only 8 bytes of process data information in maximum, so the size indicator `Length` fixes the number of valid byte in this PDO and defines herewith the number of bytes to copy from and into the process data area of the dual-port memory. The highest bit in the length indicator byte has assigned a specail function. It defines if the PDO data is either handled as word oriented data or byte oriented one. If word oriented is choosen, the behaviour of the word data interpretation in this PDO can be changed between LSB/MSB and vice versa. This can be done in the bus parameter data set globally for all word oriented PDOs.

```
#define MSK_LENGTH    0x7f
#define MSK_WORD_PDO 0x80
```

Choosing word oriented handling makes only sense if the configured PDO data is larger than 1 byte.

The COB-ID defines the CAN identifier, the node sends or receives this PDO.
The CiA draft specification defines several transmission types for PDO. The value in `bTransType` corresponds to the CiA draft specification. The SYNC mechanism will be activated only if at least one node uses SYNC controlled transmission types.

One entry in the PDO list results a corresponding entry in the `ADDTab`. In this table the offset address in the dual-port memory of each PDO is held down. See the following structure:

Header Structure	variable name	type	explanation
COM_ND_PRM_ADD_TAB	usAddTabLen	word	Length of the following add. tab data inclusive the length of the size indicator
	bInputCount	byte	number of inputs following in the IO_Offset table
	bOutputCount	byte	number of output following in the IO_Offset table
	ausIOOffsets[...]	word array	byte IO_Offset in the order: first all input offsets then all output offsets

If for example a node has one RXPDO and one TXPDO, the value for `bInputCount` and the value for `bOutputCount` must be set to 1 followed first by the word input offset address and then the word output offset address. The length indicator `usAddTabLen` in this case is 8.
If a node has more than one of the same type of PDO, first all input offset addresses must be set in the `ausIOOffsets` table and than all output offsets can follow.

5.2.5.1.2 Download Message of the Node Parameter Data Set

Download example of a node parameter data set with the address 4, without using the sequenced download procedure.

command message			
variable	type	value	signification
msg.rx	byte	3	receiver = user at HOST
msg.tx	byte	16	transmitter = CAN-Task
msg.ln	byte	0-240	length of message
msg.nr	byte	j	number of the message
msg.a	byte	0	no answer
msg.f	byte	0	error, status
msg.b	byte	68	command = COM_Download
msg.e	byte	0	extension, not used
msg.d[0]	byte	0	Req_Adr, not used
msg.d[1]	byte	4	Area_Code = node address
msg.d[2]	word	0	Add_Offset, 0 = beginning of the internal buffer
msg.d[4]	word	???	Node_Para_Len = length of the following data set + 2
msg.d[6]	byte	128	Nd_Flag = set node active
msg.d[7]	word	???	Add_Type_Info, see CAL draft specification
msg.d[9]	word	???	Dev_Prof_Number, see CAL draft specfication
msg.d[11]	word	1-2048	COB_ID_EMCY
msg.d[13]	word	1-2048	COB_ID_GUARD
msg.d[15]	word	1-65535	Guard_Time in multiples of 1msec
msg.d[17]	byte	1-255	life time factor
msg.d[18]	2 bytes	0	Octet1 - Octet2 (reserved)
msg.d[20]	word	???	Sdo_Cfg_Data_Len, size of the following Sdo_Data field + 2
msg.d[22]	structure	CAN_SDO_CFG_DATA	Sdo_Cfg_Data
msg.d[...]	word	???	Pdo_Cfg_Data_Len, size of the following Pdo_Data field + 2
msg.d[...]	structure	CAN_PDO_DATA	Pdo_Data
msg.d[...]	word	???	Add_Tab_Len, length of following Add_tab_table + 2
msg.d[...]	byte	???	Input_count, number of following input offsets
msg.d[...]	byte	???	Output_count, numder of following output offsets
msg.d[...]	word array	???	IO_Offsets[...], byte offsets in the dual-port memory where to locate the data

The structure of the node parameters in this message could not be laid down static, because the dynamic structure mechanism of one node data set causes different lengths of the several structure elements. Therefore no obvious addresses can be fixed to the several start addresses of the different parameters.

See below the corresponding structures in the header file:

```
COM_ND_PRM_HEADER
COM_ND_SDO_CFG_DATA
COM_ND_PDO_CFG_DATA
COM_ND_PRM_ADD_TAB
```

5.2.5.2 The Download of the Extended Bus Parameter

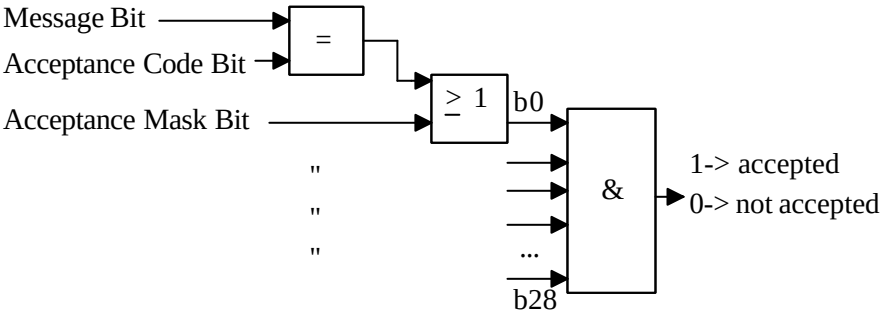
5.2.5.3 Coding of the Extended Bus Parameter Data Set

The extended bus parameter table is nesessary to download to the DEVICE for all parameter which are not covered by the standard Master Bus Parameter data set.

variable name	type	explanation
bExtendedState	byte	bit array
ulAcceptCode	long	acceptance code if device support CAN 2.0B
ulAcceptMask	long	acceptance mask if device supports CAN 2.0B
ulReserved[8]	long array	reserved area

```
typedef struct BUS_CAN_EXTtag {
  struct  {
    unsigned char bi11Bit29BitSelector: 1;
    unsigned char biNoGobalStartNode  : 1;
    unsigned char biReserved          : 6;
  } bExtendedState;
  unsigned long  ulAcceptCode;
  unsigned long  ulAcceptMask;
  unsigned char  bNodeID;
  unsigned short usProducerHeartbeatTime;
  unsigned char  bReserved;
  unsigned long  ulReserved[7];
} BUS_CAN_EXT;
```

The bit `bi11Bit29BitSelector` decides if logical 1, that the DEVICE shall support the 29Bit CAN2.0B indentifier filter logic (only for DEVICEs supporting CAN 2.0B 29bit). Together with the `ulAcceptCode` and `ulAcceptMask` it builds a filter logic for all incoming 29Bit CAN messages in the following way:



AcceptanceCode and Mask are both 32 bit long values, but the 29 rightmost bits are used for masking and codeing.

The bit `biNoGobalStartNode` decides if a global CANopen Start_Node is sent to the network by the master after having all nodes initialized. If the bit is logical 1 the master will send it, if it is logical 0 the master won't send it.

The variable `bNodeID` defines the CANopen specific Node address of the DEVICE. For the heartbeat protocol it is necessary to have an own address to calculate the COB-ID for the Heartbeat requests. The formula is COB-ID = 1792dec + `bNodeID`.

The variable `usProducerHeartbeatTime` enables or disables the Heartbeat request protocol of the DEVICE. The value can be either 0 or a multiple of a millisecond. The time itself defines the time between two consecutive requests. If a value unequal 0 is configured the heatbeat request functionality of the DEVICE is enabled. In this case it starts right after finishing the initialization when entering the state PRE-OPERATIONAL with sending the Heartbeat Requests as Producer based on the values in the variables `bNodeID` and `usProducer-HeartbeatTime`. All other node devices which support Hearbeat guarding and are configured to listen to this produced heartbeat requests can guard the DEVICE.

See below the corresponding structures in the header file:

```
BUS_CAN_EXT
```

5.2.5.3.1 Download Message of the Extended Bus Parameters

command message			
variable	type	value	signification
msg.rx	byte	3	receiver = user at HOST
msg.tx	byte	16	transmitter = CAN-Task
msg.ln	byte	45	length of message
msg.nr	byte	j	number of the message
msg.a	byte	0	no answer
msg.f	byte	0	error, status
msg.b	byte	68	command = COM_Download
msg.e	byte	0	extension, not used
msg.d[0]	byte	0	Req_Adr, not used
msg.d[1]	byte	128	Area_Code
msg.d[2]	word	0	Add_Offset
msg.d[4]	byte	0-3	bExtendedState
msg.d[5-8]	long	0-0xF8FFFFFF	acceptance code
msg.d[9-12]	long	0-0xF8FFFFFF	acceptance mask
msg.d[13]	byte	0-126	Master Node ID
msg.d[14-15]	word	0-65535	Master Producer Heartbeattime
msg.d[16-44]	long array	0	reserved area

5.2.5.4 The Download of the Bus Parameter to Start Network

5.2.5.4.1 Coding of the Bus Parameter Data Set

Downloading this bus parameter data set to the DEVICE will initially start the bus activity to the network.

variable name	type	explanation
Baud_rate	byte	fixes the baud rate
COB_ID_SYNC	word	CAN identifier for master SYNC message 1-2048
SYNC _Timer	word	Cycletime of the SYNC message send mechanism
Auto_Clear	byte	auto clear mode on / off
Intel_Motorola	byte	fixes the data format interpretation of word oriented PDO data

The parameter baudrate is a value between 0 and 8. See the following table for the several baudrate definitions:

```
#define COM_BAUD_10    8    : 10kBaud
#define COM_BAUD_20    7    : 20kBaud
#define COM_BAUD_50    6    : 50kBaud
#define COM_BAUD_100   5    : 100kBaud
#define COM_BAUD_125   4    : 125kBaud
#define COM_BAUD_250   3    : 250kBaud
#define COM_BAUD_500   2    : 500kBaud
#define COM_BAUD_800   1    : 800kBaud
#define COM_BAUD_1000  0    : 1MBaud
```

The parameter for the SYNC-message COB-ID has a valid range from 1 to 2048. This COB-ID is configured via SDO-Download function to each configured node during the bus start up process. The CANopen specific default value for the SYNC-ID is 128.
The reload timer value for the SYNC-message defines the cycle time in multiples of 1msec., when the next SYNC-message is transfered to the nodes. A value of 0 disables the timer.

The parameter 'Auto_Clear' defines the system behavior if one as active classified node has been disconnected. If 'auto clear' mode is active then the card stops the communication to all other nodes too, else it tries to restart the missed node and keeps running.

```
#define ACLR_INACTIVE 0
#define ACLR_ACTIVE     1
```

The parameter 'Intel_Motorola' changes the interpretation of word oriented PDO data from LSB/MSB to MSB/LSB and vice versa.

```
#define INTEL         0
#define MOTOROLA      1
```

See the below the corresponding structure in the header file:
`BUS_CAN`

5.2.5.4.2 Download Message of the DEVICE Bus Parameters

command message			
variable	type	value	signification
msg.rx	byte	3	receiver = user at HOST
msg.tx	byte	16	transmitter = CAN-Task
msg.ln	byte	11	length of message
msg.nr	byte	j	number of the message
msg.a	byte	0	no answer
msg.f	byte	0	error, status
msg.b	byte	68	command = COM_Download
msg.e	byte	k	extension
msg.d[0]	byte	0	Req_Adr, not used
msg.d[1]	byte	127	Area_Code
msg.d[2]	word	0	Add_Offset
msg.d[4]	byte	0-8	Baud_rate
msg.d[5]	word	1-2048	CAN identifier for SYNC-message
msg.d[7]	word	0-65535	reload timer for the SYNC message in msec
msg.d[9]	byte	0, 1	autoclear OFF = 0, ON =1
msg.d[10]	byte	0, 1	data format INTEL= 0, MOTOROLA =1

5.2.6 COM_Upload

	command message			
	variable	type	value	signification
Message header	msg.rx	byte	3	receiver = CAN-Task
	msg.tx	byte	16	transmitter = user at HOST
	msg.ln	byte	5	length of the message
	msg.nr	byte	j	number of the message
	msg.a	byte	0	no answer
	msg.f	byte	0	no error
	msg.b	byte	80	command = COM_Upload
	msg.e	byte	0	extention, not used
DNM_UPLOAD_REQUEST	msg.d[0]	byte	0	Req_Adr, unused
	msg.d[1]	byte	1-126 127	Area_Code, device address master bus parameters
	msg.d[2..3]	word	0-760	Add_Offset
	msg.d[4]	byte	1-240	Data_Len

This command allows to read back the master bus parameters or the device parameter data files.

The difference to the `COM_Download` command is, that the functions `COM_Start_Seq` and `COM_End_Seq` can not be used, but the data consistency is guaranteed anyway, but not supervised via the programmable timer algorithm.

The parameter `Area_Code` fixes the source area and can be either the master bus parameter file = 127 or a device parameter file = 0-126. The start offset within the corresponding upload area where the data will be read from must be fixed in the variable `Add_Offset`. The number of data bytes that shall be read in maximum with the service have to be inserted in the `Data_Len` variable.
If the variable `Add_Offset` plus `Data_Len` overstep the maximum actual loaded file length then the DEVICE will return no error, but return all the rest data beginning at `Add_Offset` position.

See below the corresponding structure in the header file:

```
COM_UPLOAD_REQUEST
```

answer message			
variable	type	value	signification
msg.rx	byte	16	receiver = user at HOST
msg.tx	byte	3	transmitter = DNM-Task
msg.ln	byte	x	length of message
msg.nr	byte	j	number of the message
msg.a	byte	80	answer = COM_Upload
msg.f	byte	f	error, state
msg.b	byte	0	no command
msg.e	byte	0	extention, not used
msg[0...x-1]	byte string		data of whished data set file

The `msg.d[x]` area will contain the wished data field that was addressed within the command message. I

Possible values for `msg.f` are the following:

error number msg.f	signification
0	no error
152	unknow command, DEVICE needs newer firmware
20	CON_LR, local resource not available, requested bus parameter are not present and available because card is not configured
21	CON_IV, parameter fault in request
52	CON_NI, area_code unknown
53	CON_EA, overstep of the buffer length
55	CON_IP, faulty parameter detected
57	CON_SI, sequence error
59	CON_DI, data incomplete or faulty

5.2.7 COM_Node_Diag

command message			
variable	type	value	signification
msg.rx	byte	3	receiver = user at HOST
msg.tx	byte	16	transmitter = USR_INTF-Task
msg.ln	byte	8	length of message
msg.nr	byte	j	number of the message
msg.a	byte	0	no answer
msg.f	byte	0	error, status
msg.b	byte	66	command = COM_Node_Diag
msg.e	byte	0	extension
msg.device_adr	byte	k	Rem_Adr
msg.data_area	byte	0	data area, unused
msg.data_adr	word	0	data address, unused
msg.data_idx	byte	0	data index, unused
msg.data_cnt	byte	0	data count, unused
msg.data_type	byte	5	data type byte string TASK_TDT_UINT8
msg.function	byte	1	function read TASK_TFC_READ

The command serves to read the diagnostic structure of a node. The wished address of the node must be fixed in `msg.device_adr` corresponding to the real address of the node on the bus. The execution of the command results no bus access, because the master saved the last diagnostic or emergency information of any node in an internal buffer. The status structure can be ordered anytime from the DEVICE.
The corresponding status bit of the node in the global bus statusfield in the dual-port memorys indicates, if a newly change diagnostic information within the intern status-structure of the node is available and have to be read out. So if the bit is set the user-program have to order this diagnostic information via COM_Node_Diag command and the bit will be released.

answer message			
variable	type	value	signification
msg.rx	byte	16	receiver = user at HOST
msg.tx	byte	3	transmitter = CAN-Task
msg.ln	byte	8+48	length of message
msg.nr	byte	j	number of the message
msg.a	byte	66	answer = COM_Node_Diag
msg.f	byte	0	error, state
msg.b	byte	0	no command
msg.e	byte	0	extension
msg. DeviceAdr	byte	k	Rem_Adr
msg. DataArea	byte	0	data area, unused
msg. DataAdr	word	0	data address unused
msg. DataIdx	byte	0	data index unused
msg. DataCnt	byte	48	data count = length of diagnosis structure
msg. DataType	byte	5	data type bytestring TASK_TDT_UINT8
msg. Function	byte	1	function read TASK_TFC_READ
msg.d[0]	byte	???	Nodestatus_1
msg.d[1]	word	???	Add_Info read from node OBJ1000H
msg.d[3]	word	???	Profile_Number read from node OBJ1000H
msg.d[5]	char	???	Node_state read during node guarding
msg.d[6]	char	???	Actual_Error = online error information
msg.d[7]	char	???	Emcy_Entries = number of followed emergency message entries
msg.d[8-48]	structure	???	Emcy_Data[...]

The reading of the diagnostic-information of a node causes the reset of the corresponding diagnostic bit in the 'global bus status field' of the dual-port memory. Should the remote address in `msg.device_adr` be out of range, the answer message has the error code 161. Otherwise no error is recognized and the message contains valid diagnostic data.
Every time the diagnostic field is read out, the emergency counter is reseted and the emcy_data field is filled to 0.

See below the corresponding structure in the header file:

COM_ND_DIAGNOSTICS

Octet 1: Nodestatus_1

D7	D6	D5	D4	D3	D2	D1	D0
Deact	Res	Res	Res	Guard_ Active	Prm_ Fault	Emcy_B uff_Ovfl.	No_Re sponse

D0: node not responding
D1: emergency buffer overflow
D2: difference between master and node configuration data
D3: node guarding protocol to this node is active
D4: reserved
D5: reserved
D6: reserved
D7: node is deactivated and not handled by the master

Octet 2_3: Add_info

These two byte are read out from the node whil startup. In the draft CiA specification this word is declared as extended information of the node type. For example in this word is fixed if the node supports digital input or outputs etc. .

Octet 4_5: Profile_number

These two byte are read out from the node whil startup. There are existing several predefined profile numbers each described in an own specification manual. Here is an extract of these:
Device Profile for I/O modules: 401 dec.
Device Profile for Drives and Motion Control: 402 dec.
Device Profile for Encode: 406 dec.

Octet 6: Node_State

If the node guarding protocol is active for this node, the read out node status register is written into this variable. The following value are defined in the CANopen sepcification

status value	explanation
1	disconnected
2	connecting
3	preparing
4	prepared
5	operational
127	pre-operational

Octet 7: Actual_Error

In this byte the actual online error of this node station is held down. See the table 'err_event' of the global bus status field for possible entries.

Octect8: Emcy_Entries

This byte contains the number of saved emergengy messages in the following data area.

Octet9_49: Emcy_Data

In this area the containments of the emergency messages are saved.

5.2.8 COM_SDO_Up_Download

	command message			
	variable	type	value	signification
Message header	msg.rx	byte	3	receiver = CAN-Task
	msg.tx	byte	16	transmitter = user at HOST
	msg.ln	byte	8 9-255max	length of whole message upload access download access
	msg.nr	byte	j	number of the message
	msg.a	byte	0	no answer
	msg.f	byte	0	error, status
	msg.b	byte	74	command = COM_SDO_Up_Download
	msg.e	byte	0	extension, unused
Telegram header	msg.device_adr	byte	1-127	device address, node to be addresses in the network
	msg.data_area	byte	0	data area, unused
	msg.data_adr	word	0-65535	data address, Object Index
	msg.data_idx	byte	0-255	data index, Subindex
	msg.data_cnt	byte	0-247	data count length of data to be downloaded
	msg.data_type	byte	0	data type, unused
	msg.function	byte	1 2	function code TASK_TFC_READ = upload access TASK_TFC_WRITE = download access
Data	msg.d[0-246max]	byte array	xxx	optional data in download access

This message serves to read out or to write into a definite Object of a connected node. The address of the node must be fixed in `msg.device_adr`. Its value ranges from 1 to 127. The wished object must be fixed in `msg.data_adr` and the subindex in `msg.data_idx`. To distinguish the access type the `msg.function` have to be written with the TASK_TFC_READ value for uploading read or with TASK_TFC_WRITE for downloading write access. In case of the write the command the data that shall be written have to be inserted in `msg.d[0]` area. Don't forget to assimilate the length of the message in `msg.ln` and `msg.data_cnt` then.

If the wished node is handled internally, its actual state is left for that time the command takes to handle in the network. During this time other incoming TX-PDO messages are not thrown away and are not lost, but sending new RX-PDOs to the node will be suppressed as long as the command needs to be executed. Afterwards the internal device handler retakes the old state of the node before the command execution.

	answer message			
	variable	type	value	signification
Message header	msg.rx	byte	16	receiver = user at HOST
	msg.tx	byte	3	transmitter = CAN-Task
	msg.ln	byte	8 8-255max 12	length of whole message download access upload access in case of msg.f = 3
	msg.nr	byte	j	number of the message
	msg.a	byte	74	answer = COM_SDO_Up_Download
	msg.f	byte	see table	error, state
	msg.b	byte	0	no command
	msg.e	byte	0	extension
Telegram header	msg.device_adr	byte	1-127	device address
	msg.data_area	byte	0	data area, unused
	msg.data_adr	word	0-65535	data address, Object Index
	msg.data_idx	byte	0-255	data index , Subindex
	msg.data_cnt	byte	0 0-247	data count unused in download access number of data byte in upload access
	msg.data_type	byte	0	data type, unused
	msg.function	byte	1 2	function code TASK_TFC_READ = upload access TASK_TFC_WRITE = download access
Data	msg.d[0-246max]	byte array	xxx	optional data in upload access
...or				
Abort Reason Code	msg.d[0-3]	long	xxx	Abort reason code in case of msg.f = 3

The answer message serves the HOST to get information if the wished command has been executed with or without an error. See the table on the next page for the backcoming possible error values.
If the Node itself rejects the service with an Abort SDO and the message contains the `msg.f = 3` value, then an additional error information is available in the response message at `msg.d[0-3]` position. This value is taken transparently from the Node Abort Message. The meaning of all the Abort Reason Codes are defined the CANopen Application Layer Profil or can be Node specific.

In case of an upload access the message contains the read data in the area `msg.d[xxx]`.

error number msg.f	signification
0	no error
3	service was rejected by the node with Abort SDO service. Index and subindex not valid or access right in the node does not allow the access
17	no response from the node, node is not present
19	node is not in operational state, access denied. The SDO channel is currently use by the DEVICE itself to configured the node during startup phase. Retry service again
51	maximum receive data buffer overstepped
53	fragmented protocol data oversteps receive buffer
54	unknown 'function' in HOST message or previous service still active and not confirmed to HOST application
55	node address out of range
57	sequence error in fragmented protocol, request aborted
200	DEVICE is not configured

5.2.9 COM_NMT_Module_Protocol

command message			
variable	type	value	signification
msg.rx	byte	3	receiver = CAN-Task
msg.tx	byte	16	transmitter = user at HOST
msg.ln	byte	2	length of whole message
msg.nr	byte	j	number of the message
msg.a	byte	0	no answer
msg.f	byte	0	error, status
msg.b	byte	96	command = COM_NMT_Module_Protocol
msg.e	byte	0	extension, not used
msg.d[0]	byte		NMT Command specifier
		1	Start Remote Node
		2	Stop Remote Node
		128	Enter Pre-Operational
		129	Reset Node
		130	Reset Communication
msg.d[1]	byte		NodeID
		0	address all Nodes in one service
		1-127	address specific Node

Through the NMT Module Protocol service the HOST can control the state of the NMT nodes. Normally the DEVICE itself as a master device takes care on the different states of the nodes itself, but sometimes it is necessary to control the Nodes in their states 'by hand' from the HOST application.
An NMT node is uniquely identified in the network by its Node ID, a value range of [1-127]. To address the Node in the command, place the node address in `msg.d[1]`. The value 0 is also allowed here, but addresses all nodes within the service.
The command itself has to be placed into the msg.d[0] byte of the message. Allowed code are here 1,2,128,129 and 130. These commands are standardized and specified in the CANopen Application Layer Profil.

answer message			
variable	type	value	signification
msg.rx	byte	16	receiver = HOST
msg.tx	byte	3	transmitter = CAN-Task
msg.ln	byte	0	length of whole message
msg.nr	byte	j	number of the message
msg.a	byte	96	answer = COM_NMT_Module_Protocol
msg.f	byte	0	error, status
msg.b	byte	0	no command
msg.e	byte	0	extension, not used

The answer message serves to confirm the execution of the command.

5.2.10 COM_Send_CANMsg_Transparent

5.2.10.1 DEVICEs with CAN 2.0A 11 Bit Identifier Capability

All standard Hilscher DEVICEs handle the CAN system in the CAN2.0A manner. So to send a CAN Telegram in Hilscher standard DEVICEs transparently the following message must be used:

command message			
variable	type	value	signification
msg.rx	byte	3	receiver = CAN-Task
msg.tx	byte	16	transmitter = user at HOST
msg.ln	byte	10	length of whole message
msg.nr	byte	j	number of the message
msg.a	byte	0	no answer
msg.f	byte	0	error, status
msg.b	byte	65	command = COM_Send_CANMsg_Transparent
msg.e	byte	0 2	extension answer message wished answer message suppressed
msg.d[0]	byte		Tx-Identifier-1
msg.d[1]	byte		Tx-Identifier-2
msg.d[2]	byte		Tx-Data Byte 1
msg.d[3]	byte		Tx-Data Byte 2
msg.d[4]	byte		Tx-Data Byte 3
msg.d[5]	byte		Tx-Data Byte 4
msg.d[6]	byte		Tx-Data Byte 5
msg.d[7]	byte		Tx-Data Byte 6
msg.d[8]	byte		Tx-Data Byte 7
msg.d[9]	byte		Tx-Data Byte 8

This service can be used to send a CAN message in CAN2.0A manner with 11 bits identifier transparent to the network. The DEVICE don't care the bytes `msg.[0...9]`, they are transparently copied into the used CAN-Chip queue and from there directly sent to the network.
The `msg.ln` must be set fixed to the value 10, independant of the real number of Tx data bytes that shall be sent through the following DataLenghCode value. Unused Tx-data bytes don't care in this case.
The data that shall be sent must be set in the Tx-Data `msg.d[x]` area. In maximum 8 bytes can be sent. The number of bytes in the Data field of a message is coded by the Data Length Code.

```
DataByteCount = 8*DLC.3 + 4*DLC.2 + 2*DLC.1 + DLC.0
```

For reasons of compatibility no Data Length Code > 8 soiuld be used. If a greater value then 8 is selected, 8 bytes are transmitted in the data frame with the Data length code specified in DLC

`msg.d[0]`:

D7	D6	D5	D4	D3	D2	D1	D0	Tx-Identifier 1
ID.10	ID.9	ID.8	ID.7	ID.6	ID.5	ID.4	ID.3	

`msg.d[1]`:

D7	D6	D5	D4	D3	D2	D1	D0	Tx-Identifier 2
ID.2	ID.1	ID.0	RTR	DLC.3	DLC.2	DLC.1	DLC.0	

Data Length Code

Remote Transmission Request

5.2.10.2 DEVICES with CAN 2.0B 29 Bit Identifier Capability

The DEVICEs that are supporting sending the CAN 2.0B

- TSX-CPP 100 DEVICE
- EC1 base products like CIF 80-COM and CIF 60-COM Rev#5
- optionally standard fimware version. Information if supported or not can be received at Hilscher Hotline

Of course thess DEVICE can handle the 11 bit identifier also, so sending 11 bit or 29 bit must be distinguished in two seperate messages.

5.2.10.2.1 Standard Frame 11 Bit

command message			
variable	type	value	signification
msg.rx	byte	3	receiver = CAN-Task
msg.tx	byte	16	transmitter = user at HOST
msg.ln	byte	11	length of whole message
msg.nr	byte	j	number of the message
msg.a	byte	0	no answer
msg.f	byte	0	error, status
msg.b	byte	65	command = COM_Send_CANMsg_Transparent
msg.e	byte	0 2	extension answer message wished answer message suppressed
msg.d[0]	byte		Tx Frame Info
msg.d[1]	byte		Tx-Identifier-1
msg.d[2]	byte		Tx-Identifier-2
msg.d[3]	byte		Tx-Data Byte 1
msg.d[4]	byte		Tx-Data Byte 2
msg.d[5]	byte		Tx-Data Byte 3
msg.d[6]	byte		Tx-Data Byte 4
msg.d[7]	byte		Tx-Data Byte 5
msg.d[8]	byte		Tx-Data Byte 6
msg.d[9]	byte		Tx-Data Byte 7
msg.d[10]	byte		Tx-Data Byte 8

This service can be used to send a CAN message in CAN 2.0B manner with 11 bits identifier transparent to the network. The DEVICE don't care the bytes `msg.[0...10]`, they are transparently copied into the used CAN-Chip queue and from there directly sent to the network.

The `msg.ln` must be set fixed to the value 11, independant of the real number of Tx data bytes that shall be sent through the following DataLenghCode value. Unused Tx-data bytes don't care in this case.
The data that shall be sent must be set in the Tx-Data `msg.d[x]` area. In maximum 8 bytes can be sent. The number of bytes in the Data field of a message is coded by the Data Length Code.

```
DataByteCount = 8*DLC.3 + 4*DLC.2 + 2*DLC.1 + DLC.0
```

For reasons of compatibility no Data Length Code > 8 soiuld be used. If a greater value then 8 is selected, 8 bytes are transmitted in the data frame with the Data length code specified in DLC

`msg.d[0]`:

D7	D6	D5	D4	D3	D2	D1	D0	Tx-Frame Info
0	RTR	0	0	DLC.3	DLC.2	DLC.1	DLC.0	

Data Length Code

Remote Transmission Request

Frame Format 0 = Standard Frame Format

`msg.d[1]`:

D7	D6	D5	D4	D3	D2	D1	D0	Tx-Identifier 1
ID.10	ID.9	ID.8	ID.7	ID.6	ID.5	ID.4	ID.3	

`msg.d[2]`:

D7	D6	D5	D4	D3	D2	D1	D0	Tx-Identifier 2
ID.2	ID.1	ID.0	0	0	0	0	0	

5.2.10.2.2 Extended Frame 29 Bit

command message			
variable	type	value	signification
msg.rx	byte	3	receiver = CAN-Task
msg.tx	byte	16	transmitter = user at HOST
msg.ln	byte	13	length of whole message
msg.nr	byte	j	number of the message
msg.a	byte	0	no answer
msg.f	byte	0	error, status
msg.b	byte	65	command = COM_Send_CANMsg_Transparent
msg.e	byte	 0 2	extension answer message wished answer message suppressed
msg.d[0]	byte		Tx Frame Info
msg.d[1]	byte		Tx-Identifier-1
msg.d[2]	byte		Tx-Identifier-2
msg.d[3]	byte		Tx-Identifier-3
msg.d[4]	byte		Tx-Identifier-4
msg.d[5]	byte		Tx-Data Byte 1
msg.d[6]	byte		Tx-Data Byte 2
msg.d[7]	byte		Tx-Data Byte 3
msg.d[8]	byte		Tx-Data Byte 4
msg.d[9]	byte		Tx-Data Byte 5
msg.d[10]	byte		Tx-Data Byte 6
msg.d[11]	byte		Tx-Data Byte 7
msg.d[12]	byte		Tx-Data Byte 8

This service can be used to send a CAN message in CAN 2.0B manner with 29 bits identifier transparent to the network. The DEVICE don't care the bytes `msg.[0...12]`, they are transparently copied into the used CAN-Chip queue and from there directly sent to the network.
The `msg.ln` must be set fixed to the value 13, independant of the real number of Tx data bytes that shall be sent through the following DataLenghCode value. Unused Tx-data bytes don't care in this case.
The data that shall be sent must be set in the Tx-Data `msg.d[x]` area. In maximum 8 bytes can be sent. The number of bytes in the Data field of a message is coded by the Data Length Code.

```
DataByteCount = 8*DLC.3 + 4*DLC.2 + 2*DLC.1 + DLC.0
```

For reasons of compatibility no Data Length Code > 8 soiuld be used. If a greater value then 8 is selected, 8 bytes are transmitted in the data frame with the Data length code specified in DLC

`msg.d[0]`:

D7	D6	D5	D4	D3	D2	D1	D0	Tx-Frame Info
1	RTR	0	0	DLC.3	DLC.2	DLC.1	DLC.0	

Data Length Code

Remote Transmission Request

Frame Format 1 = Extended Frame Format

`msg.d[1]`:

D7	D6	D5	D4	D3	D2	D1	D0	Tx-Identifier 1
ID.28	ID.27	ID.26	ID.25	ID.24	ID.23	ID.22	ID.21	

`msg.d[2]`:

D7	D6	D5	D4	D3	D2	D1	D0	Tx-Identifier 2
ID.20	ID.19	ID.18	ID.17	ID.16	ID.15	ID.14	ID.13	

`msg.d[3]`:

D7	D6	D5	D4	D3	D2	D1	D0	Tx-Identifier 3
ID.12	ID.11	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5	

`msg.d[4]`:

D7	D6	D5	D4	D3	D2	D1	D0	Tx-Identifier 4
ID.4	ID.3	ID.2	ID.1	ID.0	0	0	0	

answer message			
variable	type	value	signification
msg.rx	byte	16	receiver = user at HOST
msg.tx	byte	3	transmitter = CAN-Task
msg.ln	byte	0	length of whole message
msg.nr	byte	j	number of the message
msg.a	byte	65	answer COM_Send_CANMsg_Transparent
msg.f	byte	0	error, state
msg.b	byte	0	no command
msg.e	byte	0	no extension

The answer message serves to confirm that the CAN message was sent.

If the command message contains the `msg.e = 2 = RCS_NAK_MSK` then no answer message will be generated.

5.2.11 COM_Receive_CANMsg_Transparent

Receiving CAN 2.0B messages transparently is supported

- TSX-CPP 100 DEVICE
- EC1 base products like CIF 80-COM and CIF 60-COM Rev#5
- optionally standard fimware version. Information if supported or not can be received at Hilscher Hotline

5.2.11.1 Extended Frame 29 Bit

command message			
variable	type	value	signification
msg.rx	byte	16	receiver = HOST
msg.tx	byte	3	transmitter = CAN-Task
msg.ln	byte	13	length of whole message
msg.nr	byte	0	number of the message,unused
msg.a	byte	0	no answer
msg.f	byte	0	error, status
msg.b	byte	65	command = COM_Receive_CANMsg_Transparent
msg.e	byte	0	extension, unused
msg.d[0]	byte		Rx Frame Info
msg.d[1]	byte		Rx-Identifier-1
msg.d[2]	byte		Rx-Identifier-2
msg.d[3]	byte		Rx-Identifier-3
msg.d[4]	byte		Rx-Identifier-4
msg.d[5]	byte		Rx-Data Byte 1
msg.d[6]	byte		Rx-Data Byte 2
msg.d[7]	byte		Rx-Data Byte 3
msg.d[8]	byte		Rx-Data Byte 4
msg.d[9]	byte		Rx-Data Byte 5
msg.d[10]	byte		Rx-Data Byte 6
msg.d[11]	byte		Rx-Data Byte 7
msg.d[12]	byte		Rx-Data Byte 8

This command message is sent by the DEVICE if any CAN message was received fitting into the Acceptance Code/Mask filtering which both have to be set up in SyCon configuration tool.

The DEVICE don't care the bytes `msg.[0...12]`, they are transparently copied from the CAN-Chip queue into the message.

The `msg.ln` is set fixed to the value 13.

The data is set in the Tx-Data `msg.d[x]` area. In maximum 8 bytes can be received. The number of bytes in the Data field of a message is coded by the Data Length Code.

```
DataByteCount = 8*DLC.3 + 4*DLC.2 + 2*DLC.1 + DLC.0
```

`msg.d[0]`:

D7	D6	D5	D4	D3	D2	D1	D0	Rx-Frame Info
1	RTR	0	0	DLC.3	DLC.2	DLC.1	DLC.0	

Data Length Code

Remote Transmission Request

Frame Format 1 = Extended Frame Format

`msg.d[1]`:

D7	D6	D5	D4	D3	D2	D1	D0	Rx-Identifier 1
ID.28	ID.27	ID.26	ID.25	ID.24	ID.23	ID.22	ID.21	

`msg.d[2]`:

D7	D6	D5	D4	D3	D2	D1	D0	Rx-Identifier 2
ID.20	ID.19	ID.18	ID.17	ID.16	ID.15	ID.14	ID.13	

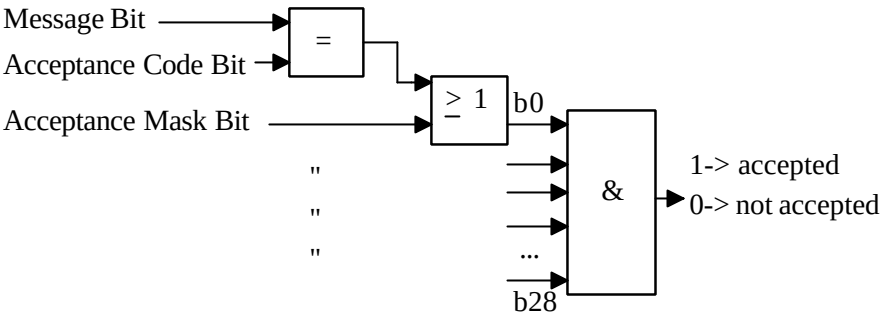
`msg.d[3]`:

D7	D6	D5	D4	D3	D2	D1	D0	Rx-Identifier 3
ID.12	ID.11	ID.10	ID.9	ID.8	ID.7	ID.6	ID.5	

`msg.d[4]`:

D7	D6	D5	D4	D3	D2	D1	D0	Rx-Identifier 4
ID.4	ID.3	ID.2	ID.1	ID.0	0	0	0	

Relation between Acceptance Code and Acceptance Mask :



For a succesful reception of a message, all single bit comparison have to signal acceptance.

5.2.12 COM_Reset_Err_Event

command message			
variable	type	value	signification
msg.rx	byte	3	receiver = CAN-Task
msg.tx	byte	16	transmitter = user at HOST
msg.ln	byte	0	data length of message
msg.nr	byte	j	number of the message
msg.a	byte	0	no answer
msg.f	byte	0	error, status
msg.b	byte	81	command = COM_Reset_Err_Event
msg.e	byte	0	extension not used

This command message serves to reset the error counters and error indicating bits in the global bus status field.

variable	type	address	short signification
global_bits	1 byte	1F40H	reset of bit TOUT and bit EVE
Bus_Error_Cnt	word	1F44H	reset to 0
Bus_Off_Cnt	word	1F46H	reset to 0
Msg_Time_Out	word	1F48H	reset to 0
Rx_Overflow	word	1F4AH	reset to 0

answer message			
variable	type	value	signification
msg.rx	byte	16	receiver = user at HOST
msg.tx	byte	3	transmitter = CAN-Task
msg.ln	byte	0	length of whole message
msg.nr	byte	j	number of the message
msg.a	byte	81	answer = COM_Reset_Err_Event
msg.f	byte	0	error, state
msg.b	byte	0	no command
msg.e	byte	0	no extension

The backcoming answer message will indicate the successful clearing of the status field. It will never contain any error indication in `msg.f`.

5.3 CAN 2.0A Layer 2 Access, Transparent Handling of CAN Messages

The CANopen master supports a full CAN layer 2 access for a transparent CAN message handling. This function works fully independant from the standard CANopen protocol that can be handled by the DEVICE in parallel. The HOST application can drive on one hand normal CANopen devices and on the other hand further non CANopen devices at the same time.

5.3.1 Sending CAN Messages at Layer 2

	command message			
	variable	type	value	signification
Message Header	msg.rx	byte	3	receiver = CAN-Task
	msg.tx	byte	16	transmitter = user at HOST
	msg.ln	byte	2...255	length of whole message
	msg.nr	byte	j	number of the message
	msg.a	byte	0	no answer
	msg.f	byte	0	error, status
	msg.b	byte	84	command = COM_Tx_CAN_Layer2
	msg.e	byte	0	extension, not used
First CAN message to be transmitted	msg.d[0]	byte		Tx-Identifier-1
	msg.d[1]	byte		Tx-Identifier-2
	msg.d[2 ...]	byte		Tx-Data Bytes if any
	...	...	...	...
Last CAN message to be transmitted	msg.d[...]	byte		Tx-Identifier-1
	msg.d[...]	byte		Tx-Identifier-2
	msg.d[...]	byte		Tx-Data Bytes if any

The command message serves to send at least one or up to 127 CAN messages transparently. After having received this message the DEVICE puts all the messages into the internal CAN-Chip queue and from there they are immediatly sent at the configured baudrate one after the other to the network. The whole message must consist of at least one CAN message which can just be the 2 bytes TX-Header-Identifier itself only. This leading identifier contains the ID and data length of the CAN message like the screen below:

D7	D6	D5	D4	D3	D2	D1	D0	Tx-Identifier 1
ID.10	ID.9	ID.8	ID.7	ID.6	ID.5	ID.4	ID.3	

D7	D6	D5	D4	D3	D2	D1	D0	Tx-Identifier 2
ID.2	ID.1	ID.0	RTR	DLC.3	DLC.2	DLC.1	DLC.0	

Data Length Code value 0 to 8

Remote Transmission Request

The Data Length Code, short DLC, represents the CAN message specific length of each CAN message. Allowed values are 0 up to 8. More data bytes than 8 can not be sent per CAN message. This is a CAN protocol limitation.
The Bit RTR is the socalled Remote Transmission Request and is always part of the CAN message. If this bit is set, no further data bytes can be sent next to the Identifier itself, but it is allowed to fill the DLC length indicator with a value from 0 to 8. This is a CAN protocol definition.

Sending all messages with header only allowes to send up tp 127 CAN message at one time. If any data byte in the CAN message has to be sent together with the TX-Identifier, it has to follow directly the identifier and the DLC parameter within the identifier has to be adjusted accordingly.

The message itself does not have any answer message.

5.3.2 Receiving CAN Messages at Layer 2

5.3.2.1 Setting up the Receive RX-Identifier CAN Filters

	command message			
	variable	type	value	signification
Message Header	msg.rx	byte	3	receiver = CAN-Task
	msg.tx	byte	16	transmitter = user at HOST
	msg.ln	byte	2...254	length of whole message
	msg.nr	byte	j	number of the message
	msg.a	byte	0	no answer
	msg.f	byte	0	error, status
	msg.b	byte	82	command = COM_CfgRx_Layer2
	msg.e	byte	0	extension, not used
First RX-CAN Filter	msg.d[0]	byte		RX-Identifier-1
	msg.d[1]	byte		RX-Identifier-2
	...	...	...	...
Last RX-CAN Filter	msg.d[...]	byte		Rx-Identifier-1
	msg.d[...]	byte		Rx-Identifier-2

This message must be sent to the DEVICE if any CAN message shall be filtered from the connected CAN network and beeing routed to the HOST application. For every CAN message an 11Bit RX-Identifier field must be set. Per message 127 RX-Filters can be set, multiple messaging is possible so that at the end the whole possible range of 0-2047 identifiers can be routed to the HOST application.
Setting the CAN filter by the identifier will always have priority against filters that are used internally for the CANopen protocol. Any ID which is filtered and was previously used by the CANopen protocol itself can not be reused for this purpose again.
Stopping the HOST communication like described in the chapter 'Starting and Stopping the Communication...' will remove all set filters.

Each 2 byte filter has the following definition:

D7	D6	D5	D4	D3	D2	D1	D0	Rx-Identifier 1
ID.7	ID.6	ID.5	ID.4	ID.3	ID.2	ID.1	ID.0	

D7	D6	D5	D4	D3	D2	D1	D0	Rx-Identifier 2
0	0	0	0	0	ID.10	ID.9	ID.8	

	answer message			
	variable	type	value	signification
Message Header	msg.rx	byte	16	receiver = user at HOST
	msg.tx	byte	3	transmitter = CAN-Task
	msg.ln	byte	0	length of whole message
	msg.nr	byte	j	number of the message
	msg.a	byte	82	answer= COM_CfgRx_Layer2
	msg.f	byte	0	error, status
	msg.b	byte		no command
	msg.e	byte	0	extension, not used

The answer message informs the HOST application that the service was executed successfully.

5.3.2.2 Indication of Received CAN Messages

	command message			
	variable	type	value	signification
Message Header	msg.rx	byte	16	receiver = user at HOST CAN-Task
	msg.tx	byte	3	transmitter = CAN-Task
	msg.ln	byte	2...254	length of whole message
	msg.nr	byte	j	number of the message
	msg.a	byte	0	no answer
	msg.f	byte	0	error, status
	msg.b	byte	83	command = COM_Rx_CAN_Layer2
	msg.e	byte	0	extension, not used
First received CAN message	msg.d[0]	byte		Rx-Identifier-1
	msg.d[1]	byte		Rx-Identifier-2
	msg.d[2 ...]	byte		Rx-Data Bytes if any
	...	...	...	...
Last received CAN message	msg.d[...]	byte		Rx-Identifier-1
	msg.d[...]	byte		Rx-Identifier-2
	msg.d[...]	byte		Rx-Data Bytes if any

This indication message is sent to the HOST application if the internal receive buffer of all the CAN messages which are configured with filters through the `COM_CfgRX_Layer2` message, contains at least one message.
The DEVICEs smallest poll rate for this buffer is 150µsec, so that at the end every 150µsec one message can be indicated. Up to 150 CAN messages can be queue in internally, if the HOST application can not handle the message in time. At 1 Mbaud and 10 bytes per received CAN message this is 150 * 111µsec (per CAN message) = 16msec. If this time is overstepped there will be a receive buffer overflow.

Each RX-Identifier in `msg.d[]` is defined as below:

D7	D6	D5	D4	D3	D2	D1	D0	Rx-Identifier 1
ID.10	ID.9	ID.8	ID.7	ID.6	ID.5	ID.4	ID.3	

D7	D6	D5	D4	D3	D2	D1	D0	Rx-Identifier 2
ID.2	ID.1	ID.10	RTR	DLC.3	DLC.2	DLC.1	DLC.0	

Data Length Code value 0 to 8

Remote Transmission Request

If the Data Length Code indicates a value unequal 0, the corresponding number of CAN message data bytes is directly following the identifier.

After getting the receive indication message, the DEVICE does not need any confirmation answer message.

6 Access of SyCons Configuration DBM-file

A DBM file is a Hilscher defined data base format file. It includes socalled tables which contain all the necessary information a DEVICE needs to start the bus communication. The DBM file itself is loaded into the internal FLASH memory of the DEVICE during the binary download procedure transparently, so that all included informations are still available after a power on reset. This binary download can be performed by a HOST application too. The procedure how to download a DBM configuration file is described in the manual DSCR_PIE.PDF in the chapter Up/Download.
Within the DBM File different data base tables can exist. They are all distinguished by an ASCII character strings of maximal 18 bytes in maximal length for identification followed by the raw structured information. Each table can have at least one data set. If for example the bus parameter shall be accessed, this table will consist of only one data set. Tables where the configuration of the network devices is held, will contain one or more records.

6.1 Creating a DBM file

The DBM file can be created based on the mother Windows-Access data base only, which is usually the SyCon configuration file itself. Of course the coded tables within the resulting DBM file will not contain every information that is held within the mother Windows-Access data base, but all the relevant information in compact format that is needed for the communication later. A DBM file can be created with selectable SyCon option File/Export/DBM.

A raw DBM file can be created too by using the dbm32.dll function DbmExportCfgToDbm() from the mother Windows-Access data base. The parameter of this function are described in the manual DSCR_PIE.PDF separatly.
The SyCon configuration program itself is using exactly the same function when perfoming the File/Export/DBM or when perfoming the binary download sequence.

6.2 Identifying the DBM file by Reading the Table GLOBAL

Each DBM file has a unique segment name character string to distinguish each DBM file for each fieldbus. This informtion is held in a global table which is part of every DBM file. The ASCII character identification string of this global table is:

HEX	47	4C	4F	42	41	4C	0
ASCII	G	L	O	B	A	L	

When reading this global table the first defined variable within DBM_SET_GLOBAL structure (see next sub-chapter) is the segment name szSegmentName. This segment name identifies the protocol the DBM file was created for. This segment name string is also available within the firmware of the DEVICE and this both character strings will be compared during download sequence of the binary DBM file .The download will be rejected if both strings do not match which means the wrong DBM file was tried to be downloaded.

6.2.1 Structure of the Table Entry GLOBAL

The table GLOBAL consists of 1 fixed structured entry. The table GLOBAL is part of every DBM file and usually the first table that can be found within the file.

```
typedef struct   DBM_SET_GLOBALtag
{
  unsigned char       szSegmentName[9];
  unsigned char       abReservedA[10];
  unsigned char       bDbmVersion;
  signed   short      sVersion;
  unsigned short      usCheckSumCRC;
  unsigned char       abReservedB[4];
  unsigned char       bSystemStart;
  unsigned char       abReservedB[3];
  unsigned short      usHostWatchdogValue;
}DBM_SET_GLOBAL;
```

The first variable szSegmentName[9] identifies the DBM file. bDbmVersion and sVersion show the version of DBM compiler that has been created the file. The usCheckSumCRC value is the checksum of the DBM file built by a CRC polynom check. The parameter bSystemStart decides on the startup behavior of the DEVICE. If the value is unequal 0 the DEVICE will wait for the startup command through the HOST application till it starts the network communication. The usHostWatchdogValue devliers the supervision time in msec the DEVICE would check for the HOSTs application activity via the cells bHostWatchdog and bDevWatchdog in the dual-port memory when started once.

6.2.2 The Segment Name within the Table GLOBAL

For the CANopen master DEVICE the segment name szSegmentName[] has the following coding:

HEX	43	41	4E	6F	70	65	6E	20
ASCII	C	A	N	o	p	e	n	

6.3 Further Table Names within the DBM file

The DBM file of the DEVICE consists of the following further tables:

HEX	42	55	53	5F	43	41	4E	0
ASCII	B	U	S	_	C	A	N	

HEX	4E	4F	44	45	53	0
ASCII	N	O	D	E	S	

HEX	50	4C	43	5F	50	41	52	41	4D	0
ASCII	P	L	C	_	P	A	R	A	M	

HEX	50	52	4F	4A	45	4B	54	0
ASCII	P	R	O	J	E	K	T	

HEX	44	45	53	43	52	49	50	54	0
ASCII	D	E	S	C	R	I	P	T	

HEX	42	55	53	5F	43	41	4E	5F	45	58	54	0
ASCII	B	U	S	_	C	A	N	_	E	X	T	

Each table name character string is terminated with a 0.

6.3.1 The Table BUS_CAN, the Bus Parameter Data Set

The table BUS_CAN will consist of one data set only. The structure is described and defined in the chapter 'Coding of the Bus Parameter Data Set' within this manual.

6.3.2 The Table NODES, Nodes Parameter Data Sets

The table NODES can consist of no record up to 127. The structure of each data set is the following:

variable name	type	explanation
Node_Address	**byte**	**Address of Node 1-127**
Node_Data_set	structure	Structure already defined in the chapter 'Coding of the Node Parameter Data Set'.

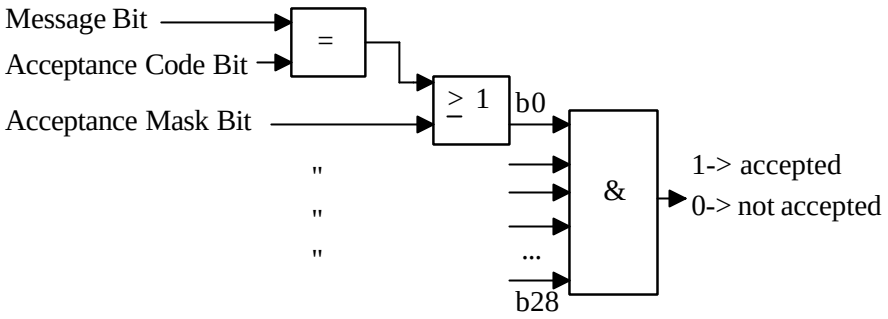
The first leading byte defines the Node Address the data set was created for. The following structure is already described and defined in this manual in chapter 'Coding of the Node Parameter Data Set'.

6.3.3 The Table BUS_CAN_EXT, Extended Bus Parameter Data Set

The table BUS_CAN_EXT will consist of one record. It has the following structure:

```
typedef struct {
  struct  {
    unsigned char bi11Bit29BitSelector: 1;
    unsigned char biNoGobalStartNode  : 1;
    unsigned char biReserved          : 6;
  } bitExtendedState;
  unsigned long  ulAcceptCode;
  unsigned long  ulAcceptMask;
  unsigned long  ulReserved[8];
} BUS_CAN_EXT;
```

The bit bi11Bit29BitSelector decides if logical 1, that the DEVICE shall support the 29Bit CAN2.0B indentifier logic. Together with the ulAcceptCode and ulAcceptMask it builds a filter logic for all incoming 29Bit CAN messages in the following way:



AcceptanceCode and Mask are both 32 bit long values, but the 29 rightmost bits are used for masking and codeing.

The bit biNoGobalStartNode decides if a global CANopen Start_Node is sent to the network by the master after having all nodes initialized. If the bit is logical 1 the master will send it, if it is logical 0 the master won't send it.

6.4 Accessing the DBM file via dbm32.dll

The dbm32.dll is normally installed with the SyCon configuration software or can be found on the system CD itself.

6.4.1 Getting the Segment Name directly with the DBM32.DLL

Use the function DbmGetFieldbusName() to get the character string of the segment name direclty. The function reads the segment fieldbus name directlty from the table GLOBAL and hand the character string over to the function caller. The function itself and its parameter are described in the manual DSCR_PIE.PDF.

6.4.2 Getting all the Table Names via the dbm32.dll

Use the function DbmGetTableDesc() to get an overview of all tables which are included in the DBM file and the corresponding number of data sets per table. Every table name is terminated with a 0 in the returned structure and for each table also a handle is returned, which must be used for later accessing to identify the table. The function is described in the manual DSCR_PIE.PDF separatly.

6.4.3 Getting the Table containments via the dbm32.dll

Use the function DbmGetRecords() now to get the whole containments of table. Use the handle hT of the table which you have received after you called DbmGetTableDesc() to identify it. If a table consist of more records, this function will return all records of this tables in one call.
After accessing the recordfile, be sure to free the allocated buffer memory with the function DbmFreeBuffer().
Use the returned value lRecordStructCount to get the whole number of record entries of the read table. Each pointer pbRecordData in the structure REC points to a new structure lRecordStructCount times.

6.5 Accessing the DBM file in Embedded HOST Applications

Embedded HOST applications normally do not have any access to the dbm32.dll file. For these the system CD contains in source code format the most important DBM functions for usage in embedded systems. The source code and header file themself are named dbm_embd.c and dbm_embd.h.

6.5.1 Function dbm_init()

This function need to be called first to fill the internal handling structure of the dbm file. The function searches the dbm file for the table names and stores them togehter with their handles in a saparate handling structure.

```
short dbm_init
    ( void * ptDbmFile )
```

Parameter:

type	parameter	description
void *	ptDbmFile	pointer to the start of the dbm file

Return values:

value	description
DBM_OK	0 = No error
DBM_DAT_DIR_FULL	3 = too much table entries found, enlarge the DBM_MAX_TAB define

6.5.2 Function dbm_get_num()

This function returns the handle of the given table name. The returned handle value is needed for further access to the table containments and is a parameter of all the other dbm functions.

```
short dbm_get_num
    ( unsigned char *pszName,
      unsigned char *pbHandle );
```

Parameter:

type	parameter	description
unsigned char *	pszName	pointer to the ASCII character name of the table
unsigned char *	pbHandle	pointer where the function shall place the handle of the wished table

Return values:

value	description
DBM_OK	0 = No error
DBM_NO_DAT	1 = table name not found within the dbm file. check the character string

6.5.3 Function dbm_get_num_of_set()

This function returns the number of data sets that are found within the given table.

```
short dbm_get_num_of_set
    ( unsigned char bHandle,
      unsigned short * pusNumberOfSets);
```

Parameter:

type	parameter	description
unsigned char	bHandle	handle of the table
unsigned short *	pusNumberOfSets	pointer where the function shall place the number of found data set entries

Return values:

value	description
DBM_OK	0 = No error
DBM_NO_DAT	1 = handle parameter not valid and out of range

6.5.4 Function dbm_fast_read()

This function serves to read out a given data set entry.

```
short dbm_fast_read
    ( unsigned char  bHandle,
      unsigned short usSetNumber,
      unsigned short usMaximumLength,
      void * pvDestinationPointer,
      unsigned short * pusRealLength );
```

Parameter:

type	parameter	description
unsigned char	bHandle	handle of the table
unsigned short *	usSetNumber	data set number that shall be read. Starting with index 1 = first data set entry.
unsigned short	usMaximumLength	maximum length of the buffer where the data set shall be copied to
void*	pvDestinationPointer	pointer to the destination buffer where the data set shall be copied to
unsigned short*	pusRealLength	pointer where the function shall place the real length of the data set

Return values:

value	description
DBM_OK	0 = No error
DBM_NO_DAT	1 = handle parameter not valid and out of range
DBM_NO_SET	2 = set number out of range
DBM_SET_DEL	5 = data set is not available
DBM_NOT_ENOUGH_FREESPACE	4 = parameter usMaximumSize indicates a too small buffer length

7 General Procedure how to get the DEVICE operative without SyCon

Like in the chapters above described, the DEVICE supports the online configuration without using the SyCon configuration tool. That means the DEVICE must be initalized in its protocol parameters first (see. chapter protocol parameters), then a warmstart must be proceeded. After that the network specific parameter must be download via message functionality `Start_Seq, Download, End_Seq`. By using these functions, the network device specific parameters must be downloaded first and then the bus parameter must follow. The download of the bus parameter is the trigger point for the DEVICE to start its network activity the first time. Remember that these download parameter aren't stored in the DEVICE FLASH memory and are lost if the DEVICE is reseted or powered down.
The just described procedure does only work, if the DEVICE isn't configured by SyCon configuration tool, else the found FLASH configuration will always have higher priority then the HOST defined configuration download parameter. To ensure that the DEVICE itself is not preconfigured by SyCon with a static FLASH configuration, for example if you have receive a new delivered one, you have to proceed the following initial sequence to get every DEVICE working:

7.1 Using Device Driver Functions

1. `DevOpenDriver()`: Enable the link of the application to the device driver
2. `DevInitBoard()`: Link application to the specific DEVICE
3. `DevPutTaskParameter()`: Set up the protocol parameter
4. `DevReset(WARMSTART)`: Execute a warm start command to DEVICE
5. `DevGetBoardInfo(GET_DRIVER_INFO)`: Read driver state
6. Examine the variable `bHostFlags` in the backcoming driver state structure
7. If `bHostFlags` indicates the bits `RDY` and `RUN` then the DEVICE is configured by SyCon. Then execute Delete database message to DEVICE by using `DevPutMessage()` and `DevGetMessage()` procedure. Goto step 3 again
8. Use now `DevPutMessage()` and `DevGetMessage()` procedure to download the network specific configuration. After the download of the bus parameter data set the DEVICE automatically starts up the network.

7.2 Using direct access to the dual-port memory

1. Examine the cell `bHostFlags` directly. If cell indicates the bits `RDY` and `RUN` then the DEVICE is configured by SyCon. Then execute `delete database` message (see chapter in this manual) to DEVICE by using corresponding message algorithm described in the `toolkit general definitions` manual. Goto step 1. If cell indicates `RDY` only then goto step 2.
2. Write protocol specific parameter into corresponding `Task2Parameter` area.
3. Write WARMSTART = 0x40 into cell `bDevFlags` to execute the DEVICE's warmstart.
4. Now download the network specific configuration via message procedure. After the download of the bus parameter data set the DEVICE automatically starts up the network.

8 Appendix

8.1 Technical Compliance to Specification DS 301 Version 4.01

Summary of supported Services:

- Physical Layer
- CAN Frame types 11 Bit, optional 29 Bit (only customer specific)
- Read/Write PDO Protocol
- Trigger for PDO transmission, event driven and remotly requested, transmission modes synchronous and asynchronous.
- PDO Mapping, in accordance DS 301, V3.0 and V4.01
- Emergency message collection
- NMT Functionality, supporting Reset_Communication, Start_Remote_Node, Stop_Remote_Node and Enter_Preoperational.
- SYNC Protocol
- Node Guarding Protocol
- Optional Heartbeat Protocol
- SDO Up/download Protocol
- Simple Boot-Up process, reading object 1000H for identifying

Not Supported Services:

- Bootup Protocol, no reaction on Bootup Events of a Node.
- Optional High Resolution Synchronization Protocol
- Optional SDO Block Up/Download Protocol
- Timer Triggered PDO transmission
- Dynamic SDO establishment
- Multiplexed PDOs, MPDO Protocol
- Reading Identity Object of a Node for identification
- TIME Protocol
- Verify Configuration Boot-Up Process
- Boot up Process in accordance DS 302, NMT Master
- Coexistence of a second NMT Master
- Modular Devices