Manual

CANopen
Master API for Windows

Software package for the development of CANopen Master applications under Windows

Software Version 6.1

HMS Technology Center Ravensburg GmbH
Helmut-Vetter-Straße 2
88213 Ravensburg
Germany

Tel.: +49 751 56146-0
Fax: +49 751 56146-29
Internet: www.hms-networks.de
e-Mail: info-ravensburg@hms-networks.de

Support

In case of unsolvable problems with this product or other products please contact our support in written form by:

Fax: +49 751 56146-29
e-Mail: support@ixxat.de

Further international support contacts can be found on our webpage www.hms-networks.de

Copyright

Registered Trademarks

Document number: 4.12.0132.20000
Version: 6.5

1 Introduction

The CANopen Master Application Programming Interface (API) is a programming library for connecting a PC-application to a CANopen-network in form of a single monolithic file named XatCOP60.dll

This functionality comprises the availability of process data objects (**PDO**s) and service data objects (**SDO**s) for the direct information exchange with individual network subscribers and the provision of system services of the synchronization object (**Sync**) and of the central time object (**Time Stamp**). In addition the device error messages (**Emergency messages**) can be evaluated. For the network, so-called NMT-services and node-monitoring mechanisms are available.

In particular the following features of the CANopen specification CiA-301 V4 are supported:

- SDO block transfer
- Heartbeat mechanism
- Bootup message
- Sync counter

With these functions the CAN-identifiers are allocated according to the so-called **Predefined Connection Set**. For PDOs and SDOs, however, other CAN-identifiers can also be defined.

The user must be familiar with the various mechanisms and terms of CANopen. More information can be obtained from the relevant specifications (CiA-301, CiA-401, ...), which are available from CiA (`www.can-cia.org`).

A detailed introduction to CANopen is given in the book: K. Etschberger, "Controller Area Network, Basics, Protocols, Chips and Applications", 2001 IXXAT Press, ISBN 3-00-007376-0.

Other current information on the software not included in the manual is available in the form of `README`-files on the data carriers supplied. Please therefore check whether `README`-files are contained on the diskette/CD.

1.1 Where to find what

This manual describes all functions and data structures provided by CANopen Master API for controlling a CANopen-system.

Section 3 provides an overview of the software. The typical situations and processes for the use of the CANopen Master API are briefly described under 'Application examples' (section 4). Section 5 'Individual functions' contains the complete function reference.

This documentation is then concluded with the appendices, which describe the internal command interface of the underlying CANopen Master firmware, the performance parameters and the scope of delivery as well as a section on typical sources of errors and stumbling blocks from practical experience.

1.2 Basic specifications

/1/ CiA-301 CANopen Application Layer and Communication Profile

/2/ CiA-302 Additional application layer functions

/3/ CiA-305 Layer Setting Services and Protocol (LSS)

1.3 Definitions, acronyms, abbreviations

Bootup Message

Special case of a →Heartbeat message. With this the node signals the successful transition to the "pre-operational" state. See also NMT.

CAN-ID

The CAN-ID or message-identifier identifiers a CAN-message and at the same time defines the message priority. The highest priority ID 0 is reserved for network management services →NMT

CiA

CAN in Automation e.V.: Organisation of CAN-Bus device manufacturers and users.

Client-SDO

A Client-SDO is understood as the initiator of an SDO-transmission. It has access to the object dictionary entries of an "SDO-Server". →SDO, OD

Communication Cycle Period

Communication Cycle Period defines the time interval between sequential →Sync-objects.

Communication parameters

The attributes of a →PDO are described in the communication parameters. The attributes include →Transmission Type, →Inhibit Time and of course the →CAN-ID.

Emergency Message

With a high-priority Emergency object a device signals the occurrence of a fatal internal device error or the reset of one or all device-internal errors.

Guard Time

The NMT-Master cyclically transmits a request to the NMT-Slave to send its current node state. This request must be answered within the node lifetime. The node lifetime of a node is given by Lifetime Factor ∗ Guard Time of the node. The NMT-Slave does not carry out guarding of the NMT-Master if the Guard Time is parameterized with 0. However, the Node Guarding protocol is answered. The reactions to violations of the Node Guarding are described in the CANopen specification.

Heartbeat

Independent cyclical transmission of the node state by means of a so-called Heartbeat message. Alternative to →Node Guarding by means of →Guard Time. It reduces the bus load due to absence of the request message.

Inhibit Time

A Process Data Object (→PDO) may only be transmitted again after expiry of this time.

LSS: Layer Setting Services

Service element of the application layer, which enables the setting of basic device-communication parameters such as baudrate and →Node-ID.

NMT: Network Management

Service element of the application layer in the CAN reference model, which comprises the configuration, initialization and error processing in the network as well as network-wide process synchronisation. CANopen recognizes four main states: "initialization", "pre-operational", "operational" and "stopped". NMT-commands trigger the state transition of a CANopen node. The Network Management has a Master-Slave structure.

Node Guarding

Cyclic guarding of a node. Node Guarding can be implemented either by means of →Guard Time or via →Heartbeat.

Node-ID

An individual device in the CAN-network is unmistakeably defined by its node number (between 1 and 127). This node number is used by →Predefined Connection Set for the pre-defined identifier allocation. Node-ID 0 is reserved for →NMT-services.

OD: Object Dictionary

The object dictionary is a data structure via which all objects of a CANopen device can be addressed. The object dictionary is subdivided into an area with general information on the device, an area containing the communication parameters and an area that describes the specific device functionality. The data in the OD are addressed via an index and a subindex. Via the entries (objects) of the object dictionary, the application objects of a device such as input and output signals, device parameters, function or network variables in standardized form are made accessible via the network. The object dictionary forms the interface between network and application process.

PDO: Process Data Object

PDOs represent the actual means of transport for the transmission of process data. A PDO is transmitted by a "Producer" and can be received by one or more "Consumers". The process data transmitted by a Producer in a PDO can contain a maximum of 8 bytes. A PDO is transmitted without acknowledgement and requires an identifier unmistakeably allocated to the PDO. The meaning of the transmitted data is defined by the identifier used and the PDO-mapping allocated to a PDO. The communication-specific parameters define the mode of the PDO. For the management of PDOs, both PDO-Producer and PDO-Consumer require appropriate data structures. The PDO-Producer manages the data it requires in the form of so-called TPDO data structures, the data to be received by a PDO-Consumer are managed by so-called RPDO data structures.

Predefined Connection Set

Preset allocation of the →CAN-ID based on the →Node-ID and on the function code. The 127 nodes are differentiated via the lower valued bits of the identifier. For the following communication objects Predefined Connection Set predefines the CAN-ID: Node Guarding/Heartbeat, Emergency Message, Sync Object, Time Stamp, Server-SDO 1, RPDO 1 to 4 and TPDO 1 to 4.

RPDO: receive-PDO

→PDO

SDO: Service Data Object

An SDO is a CAN-communication object used for initialization and parameterization of CANopen devices or for transmission of long data records. SDOs are used for read or write access to the entries in the object dictionary of a device. An entry is accessed by stating the index and subindex.

SDO Timeout

An SDO-request must be answered within the timeout time.

Server-SDO

Each device must support at least one server-SDO and thus allow access to the entries in its object dictionary. The specification of an SDO-Server-object requires the definition of one CAN-identifier each per transmission direction

(acknowledged service), and specification of the corresponding Client- or Server-node if the dynamic structure of SDO-connections is supported.

Sync-object

Sync-object is used for synchronized data collection, synchronized command-strobing and cyclic transmission of process data. The receipt of a Sync-object triggers updating and transmission of synchronous messages. For this, a device (Sync-Producer) cyclically transmits the high-priority Sync-object. The Sync-object is only completely described with specification of the →Communication Cycle Period parameter and of the →Synchronous Window Length parameter. If a parameter is initialized with 0, it has no effect.

Synchronous Window Length

Time window after a →Sync-object within which the →PDOs must be transmitted, which have synchronous transmission type.

Time-Stamp-Message

This is used for resynchronization of the local timers, to ensure higher requirements of synchronicity.

Transmission Type

The mode of a →PDO is specified via the transmission type in the communication profile of a device. CANopen provides the following transmission types for PDOs:

synchronous: the transmission is, depending on a Sync-object

either acyclic: once

or cyclic: with each receipt or after a number of Sync-objects specifiable with the →Transmission Rate.

asynchronous: the transmission is triggered by a manufacturer-specific or by an event defined via the device profile.

Remote: the transmission occurs only after being requested by another subscriber (PDO Consumer).

Transmission Rate

For the cyclic-synchronous mode of a →PDO, the value of the Transmission Rate represents the number of Sync-objects that must have been received until the PDO is transmitted again.

TPDO: transmit-PDO

→PDO

1.4 Typographical conventions

The following typographical conventions apply to this handbook.

Type	Meaning
V20_CN32.EDS	User inputs or operating system-specific elements such as file names etc.
Baudrate	Lettering of a windows control or screen output
TPDO	CANopen term

1.5 Support

For additional information on IXXAT products, FAQ lists and installation tips, please refer to the support section of the IXXAT website (www.ixxat.com), which also contains information on current product versions and available updates.

If you have any further questions after studying the information on our website and the manuals, please contact our support department. The support section on our website contains the relevant forms for your support request. In order to facilitate our support work and enable a fast response, please provide precise information on the individual points and describe your question or problem in detail.

If you would prefer to contact our support department by phone, please also send a support request via our website first, so that our support department has the relevant information available.

1.6 Return of defect Hardware

If it is necessary to return hardware, please download the relevant RMA form from our website and follow the instructions on this form.

In the case of repairs, please also describe the problem or fault in detail on the RMA form. This will enable us to carry out the repair quickly.

2 Commissioning

2.1 System requirements

The software is available for Windows XP, Windows Vista and also Windows 7/8/10 in 32bit as well as in 64bit.

	Minimum requirement
Operating system	Windows XP Professional SP2
Processor	Pentium4 1,3 GHz
Main memory	512 MB RAM

The system requirements are mainly determined by the Client-application, since the core functionality of the CANopen Master API runs directly on the CAN-board independently of the computer.

2.2 Supported CAN-boards

The CANopen Master API primarily works together with so-called active CAN-boards. These have a microcontroller that executes the CANopen Master Firmware contained in the API. If the CAN board is equipped with a 16-bit microcontroller, all available CAN lines can be used independently of one another.

In this way the following IXXAT CAN interfaces are supported. More detailed information on the individual capacity limits is given in Appendix B - Performance characteristics.

PC Interface	CAN Interface	Order no.	Microcontroller
ISA	iPC-I 320 iPC-I 320 / 104	1.01.0040 1.01.0043	Dallas 80C320
PCI	iPC-I 320 / PCI	1.01.0044 1.01.0039	Dallas 80C320
PCI	iPC-I XC16 / PCI iPC-I XC16 / PMC	1.01.0047 1.01.0049	Infineon XC161
PCI-Express	iPC-I XC16 / PCIe	1.01.0053	Infineon XC161
PCI-Express	CAN-IB200 / PCIe	1.01.0233 1.01.0234	NIOS II
PC-Card / PCMCIA	tinCAN V4	1.01.0028	Dallas 80C320
PC-Card / PCMCIA	tinCAN161	1.01.0026	Infineon C161

USB	USB-to-CAN II	1.01.0062	Infineon C161U
USB	USB-to-CAN compact	1.01.0087 1.01.0088	Infineon C161U
Ethernet	CAN@net II / VCI	1.01.0086 .10200	Freescale MCF5235

Furthermore, there is a windows library included named XatCOP60_VCI3.dll that allows for using all other IXXAT CAN Interface boards such as USB-to-CAN V2 (1.01.0281) based on VCI3. This way CANopen Master firmware is running as thread in user mode of the Windows operating system. So it is able to benefit from the internal clock of the system processor, on the other hand, it is also subjected to the restraints of the operating system kernel which is to be known as not being real time capable. The latter makes itself felt, among other things, by variations of the sync cycle time. Choosing of the variant, i.e. the selection of either the microcontroller specific firmware or the generic VCI3 firmware is automatic. This means that the microcontroller variant is always preferred, unless there is none for the desired CAN board. In terms of programming of your client application there is no difference between both the variants. The application programming interface as well as the scope of operation is identical.

Make sure the aforementioned DLL is located in the same folder as the CANopen Master API DLL itself, otherwise you will get the error BER_k_BOARD_DLD_ERR when calling the API-function COP_InitBoard().

Access to the CAN-Bus is only possible with IXXAT CAN-interface boards. For installation of the CAN-interface board used, please refer to the manuals included with the hardware.

2.3 Before installation

The CANopen Master API installation is depending on the VCI driver software. This has the advantage that the CANopen Master API is independent of operating system and hardware. The availability of the CANopen Master API for a certain operating system/hardware combination therefore depends only on the VCI support. For an up-to-date overview of the VCI version(s) required for the CANopen Master API, please consult the IXXAT website at `www.ixxat.com`. For downloading of the VCI driver package see the download area on the homepage `www.ixxat.com`

The driver of the CAN boards must be installed *before* the CANopen Master API, otherwise you will get the error BER_k_VCI_INST_ERR when calling the API-function COP_COP_InitBoard ().

2.4 The actual installation

Start the file `setup61.exe` found on the CD supplied and follow the instructions of the program. You must normally have administrator rights to be able to install the software successfully.

2.5 Getting Acquainted with the API

There is one identical sample program each for the programming languages Microsoft C#, Microsoft Visual C++, Borland Delphi, Borland C++ Builder and Microsoft Visual Basic.NET. The sample program shows how to initialize a CANopen network and to address an I/O-node with a digital 8-bit input/output. To find the commands relevant for CANopen, you can search for the abbreviation `COP_`. By clicking the button **Init CANopen-Master**, the following procedure is followed.

(1) `COP_InitBoard` registers the CAN-board with the CANopen Master API.

(2) `COP_InitInterface` initializes the CANopen Master Firmware and transfers the baudrate.

(3) With `COP_AddNode` the I/O-node is registered with the Master API using Node Guarding.

(4) With `COP_SearchNode` it is checked whether the node really exists in the network.

(5) `COP_CreatePDO` is used to configure one **Process Data Object PDO** for the input and output.

(6) Now the node is started with `COP_StartNode` and a timer for polling is started.

(7) The timer function reads out the **RPDO** with `COP_ReadPDO`.

(8) The node outputs are written to with `COP_WritePDO`.

(9) Finally the network is set to standby (**Pre-Operational**) with `COP_EnterPreOperational`.

Figure 2-1 shows this typical sequence of the CANopen Master API function calls in a Windows application with use of process data objects and service data objects.

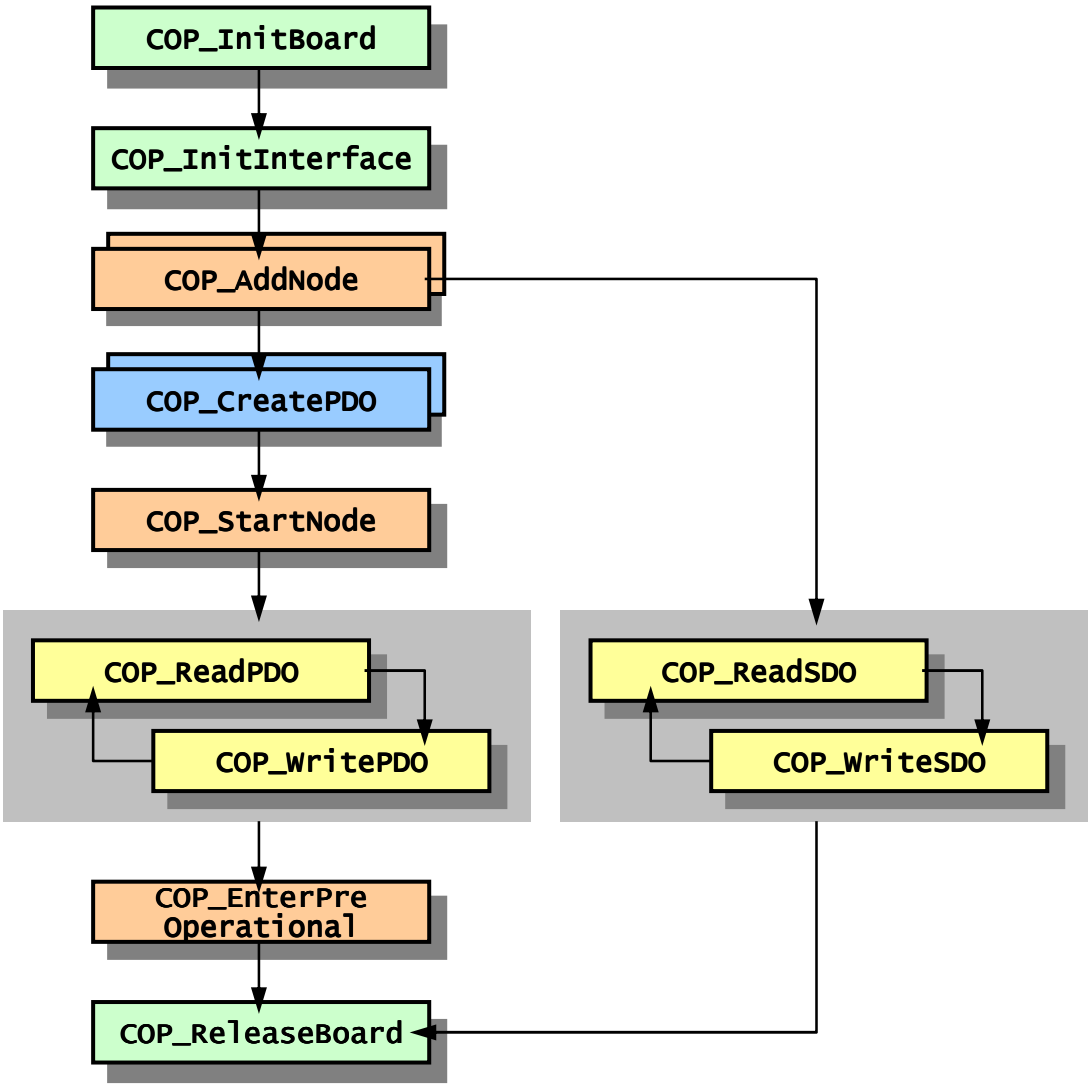

Fig. 2-1: Typical sequence of function calls

3 Overview

The CANopen Master API provides functions for the CANopen network management and the CANopen object communication.

Independent CANopen Master Firmware runs on the microcontroller of the IXXAT CAN-board, which independently configures and intensively uses the CAN-Controller and the local RAM-memory. For communication with this firmware two command queues (CMD-queues) are used and for communication with the network six data queues.

The CANopen Master API DLL loads the firmware into the volatile memory on the board, sets up the queues and handles editing of the data for the queues.

All commands for CANopen Master Firmware on the board are exchanged via the two command queues. For fast exchange of communication and emergency objects the six data queues are used (Figure 3-1). In the multi-line firmware the queues are available several times, so that complete independence of the CAN lines is ensured.

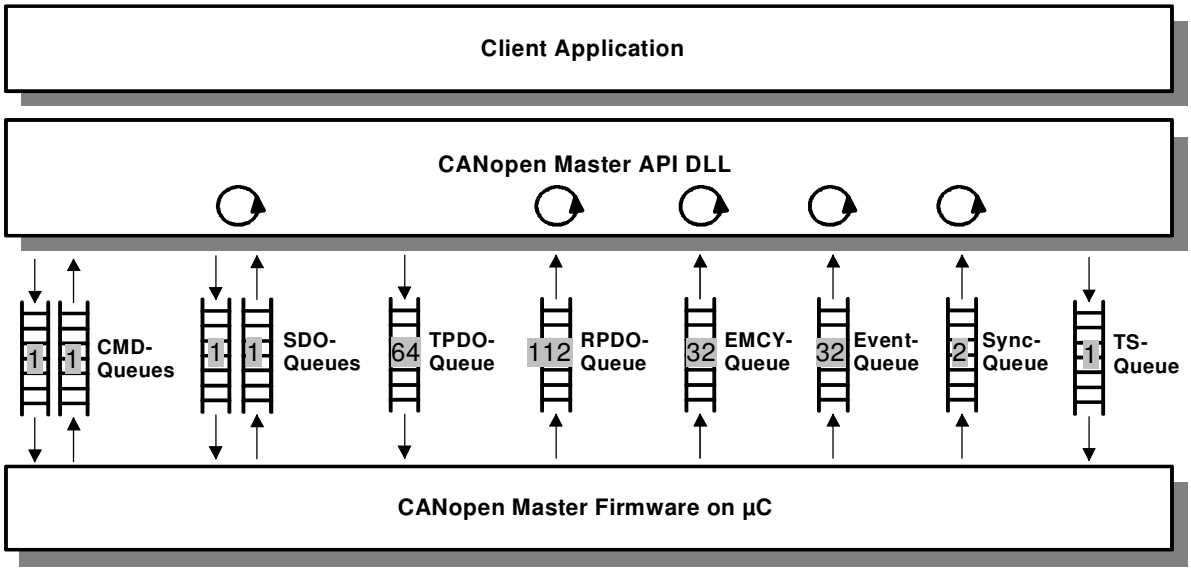

Fig. 3-1 Communication between Master API DLL and Master Firmware (single-line variant)

The firmware is configured and parameterized in conversational mode via the command queues. Almost all functions of the CANopen Master API are edited to operation structures within the DLL, transfered to the firmware and then processing resp. confirmation by the firmware is awaited (see also section 3.2).

The external CANopen devices are configured with the aid of the two SDO-queues. API-functions are available for asynchronous SDO-transfer (not blocking the Windows application) and synchronous SDO-transfer (blocking the Windows application).

The process data exchange between the Windows application and the CANopen network takes place by means of the PDO-transmit- and PDO-receive-queue.

Finally there are three other receive-queues by means of which the Windows application can be informed of emergency objects, net-events and Sync-objects as required. A Callback or a Message-ID can be assigned to each of these queues, so that when the corresponding CANopen message is received, a function is called directly inside the Windows application or a Message is posted to an application window or a worker thread.

3.1 Function categories

The functions of the CANopen-Master API can be divided into the following categories. The complete reference is given in section 5.

3.1.1 Basic API-functions

They are used for initialization and parameterization of the API, selection of the CAN-board and for communication with the Master Firmware on the CAN-board.

- ***Selection of the CAN-board***
  - COP_InitBoard
  - COP_ReleaseBoard
  - COP_GetBoardInfo

- ***Initialization and parameterization of the API***
  - COP_InitInterface
  - COP_DefineCallbacks
  - COP_DefineMsgRPDO
  - COP_DefineMsgEvent
  - COP_DefineMsgEmergency
  - COP_DefineMsgSync
  - COP_GetThreadIds
  - COP_Reset_DLL

- ***Communication with the CANopen Master Firmware***
  - COP_SendMsg
  - COP_GetMsg
  - COP_SetCommTimeOut
  - COP_GetStatus
  - COP_TestCommand

3.1.2 Functions for the network management

These are functions for setting up the network and controlling the individual CANopen nodes.

- ***Setting up the CANopen network***
  - COP_AddNode
  - COP_DeleteNode
  - COP_SearchNode
  - COP_GetNodeInfo
  - COP_ChangeNodeParameter
  - COP_SetEmcyIdentifier
  - COP_ConfigFlyMaster
  - COP_StartFlyMaster
  - COP_GetStatusFlyMasterNeg

- ***Controlling the individual CANopen nodes***
  - COP_StartNode
  - COP_StopNode
  - COP_ResetComm
  - COP_ResetNode
  - COP_EnterPreOperational
  - COP_GetNodeState

3.1.3 CANopen-object management

For creating and parameterizing CANopen communication objects.

- ***Process data objects (PDOs)***
  - COP_CreatePDO
  - COP_DeletePDO
  - COP_GetPDOInfo

- ***Service data objects (SDOs)***
  - COP_CreateSDO
  - COP_GetSDOInfo
  - COP_SetSDOTimeOut

- ***System services***
  - COP_DefSyncObj
  - COP_SetSyncDivisor
  - COP_GetSyncInfo
  - COP_EnableSync
  - COP_DisableSync
  - COP_InitTimeStampObj
  - COP_StartStopTSObj
  - COP_GetTimeStampObj

3.1.4 CANopen communication

For the direct information exchange with the individual CANopen devices.

- ***Process data objects (PDOs)***
  - COP_ReadPDO
  - COP_ReadPDO_S
  - COP_RequestPDO
  - COP_WritePDO
  - COP_WritePDO_S

- ***Service data objects (SDOs)***
  - COP_ReadSDO
  - COP_WriteSDO
  - COP_PutSDO
  - COP_GetSDO
  - COP_CancelSDO

- ***System services***
  - COP_GetEmergencyObj
  - COP_GetEmergencyObj_S
  - COP_CheckSync
  - COP_GetEvent

3.1.5 LMT services

- COP_LMT_ConfigNode
- COP_LMT_GetAddress
- COP_LMT_ConfigModuleID
- COP_LMT_IdentifyRemoteSlaves

3.1.6 LSS services

- COP_SetLSSTimeOut
- COP_LSS_InquireAddress
- COP_LSS_InquireNodeID
- COP_LSS_ConfigNodeID
- COP_LSS_ConfigBitTiming
- COP_LSS_ActivateBitTiming
- COP_LSS_IdentifyRemoteSlaves
- COP_LSS_IdentifyNonConfRemoteSlaves
- COP_LSS_Fastscan

3.2 Internal use of the command queues

Almost all functions of the CANopen Master API work internally according to a fixed plan with COP_SendMsg() and COP_GetMsg(). This plan is shown as an example in Figure 3-2.

First the command record COP_t_QueueMessage is initialized according to the DLL-function called by the application program, provided with the command-Opcode and filled with the function parameters.

This command record is entered in the transmit-command queue by calling COP_SendMsg(). Then the acknowledgement (confirmation) of the Master Firmware is awaited by cyclically querying the receive-command queue restricted by time by calling COP_GetMsg().

The error code is extracted from the received acknowledgement command record and returned to the application program.

As these two dialog functions are also available to the application program, an own implementation of the CANopen Master API DLL functionality can be written if required. The data structures and command opcodes required for this are contained in the file copcmd. Detailed information is given in Appendix D - Data structures of the command queues.

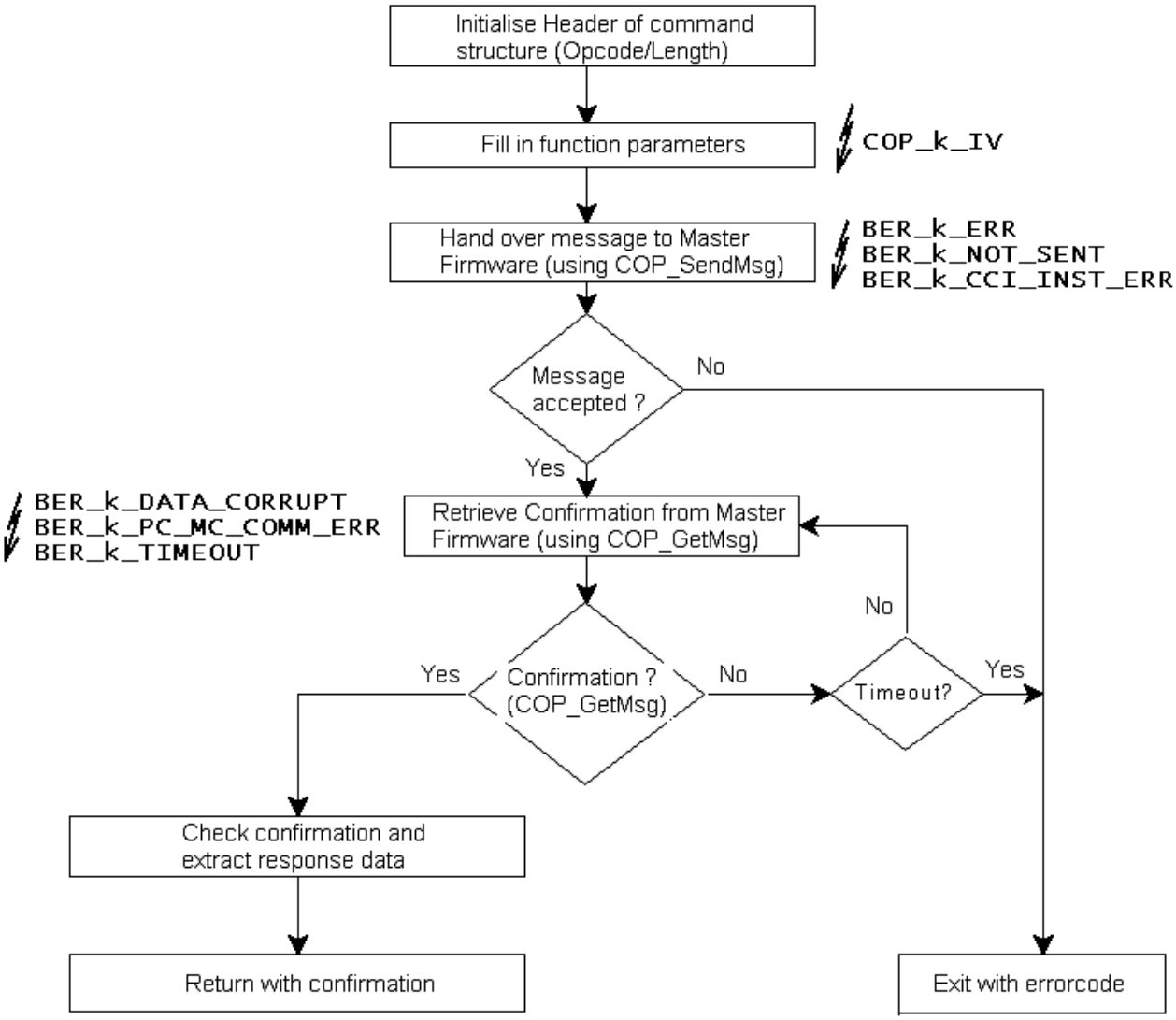

Fig. 3-2: Flowchart of a standard request

4 Application examples

To gain a feeling for the use of the CANopen Master API, this section shows some typical tasks. For reasons of clarity, error processing is not discussed.

4.1 Sample programs supplied

There is one simple identical sample program `DigIODemo.exe` each for the programming languages Microsoft C#, Microsoft Visual C++, Borland Delphi, Borland C++ Builder and Microsoft Visual Basic.NET. To use it correctly, an I/O-node standardized in accordance with CiA-401 must be available. The node should have 8 digital inputs and outputs each. In addition the **CAN-identifier**s used must be set according to the **Predefined Connection Set**. With this a simple CANopen network consisting of an I/O-Slave and the CANopen Master API can then be constructed.

The condition for the perfect functioning of the sample programs is the successful installation of the CANopen Master API. Then the sample programs are contained in the subdirectory `SAMPLES` of the Public Documents folder.

4.1.1 Calling sample programs

The sample programs expect the following command line parameters:

<Boardtype>: This parameter defines which CAN-board shall be used for the CANopen Master API.
The following board names are accepted: `IB200`, `USBV2`, `PCI320`, `XC16`, `TINCAN161` and `CAN@NET`.
The easiest way is to enter `SELECT` here. Then a board selection dialog is opened in which you can select your board.

<Baudrate>: Baudrate used in kBit/s.

<Node>: The node-ID of the external I/O-node.

<CANline>: Required CAN line (`0`, `1`, `2` or `3`) or `–1` for single-line firmware of an active CAN-board (see section 2.2).

4.1.2 Structure of the sample programs

The sample programs consist of a graphic interface with which the states of the inputs or outputs of the I/O-node are shown and manipulated. The sample programs roughly follow the procedure described in section 4.4. For further information, see section2.5.

4.2 Reading an object dictionary entry via SDO

In addition to the PDO functionality described in the previous section, reading an object dictionary entry with the CANopen Master API is now described.

(1) Initialization of the CAN-board with `COP_InitBoard()`
The board type (`pBoardtype`) and the unique Boardidentifier (`pBoardID`) as well as the required CAN line on the relevant board are transfered. If the call is successful, a Boardhandle (`pBoardhdl`) is returned that identifies the board with all subsequent API-calls.

(2) `COP_InitInterface()` initializes the CANopen Master Firmware.
Here the baudrate of the network (`baudrate`) and the node-ID of the CANopen Master Firmware itself (`node_no`) are set. The node-ID is only relevant for use of the Heartbeat mechanism.

(3) The target node is registered with the Master Firmware with `COP_AddNode()`. To be able to work with a node in the network, it must always be registered with the firmware first. For this internal management structures are allocated, its mandatory SDO-channel (**Server-SDO 1**) is being established, and status variables are set.
With node registration, the following can be set: node-ID and node monitoring mechanism (**Guarding** or **Heartbeat**).

(4) Now the CAN- Identifier (**CAN-ID**), for example, can be read out from the object dictionary for the first receive-process-data-object (**RPDO**) with `COP_ReadSDO()`.
This information is given according to the CANopen-specification DS-301 under [1400sub01] in the node ODs. An unsigned32-value is returned with the CAN-ID. Normally the value there is set according to the **Predefined Connection Set** (i.e. CAN-ID = 0x200 + node-ID). The basic-CAN-ID-value is also found as a constant in the main header `cop` under `COP_k_S_ID_RxPDO1`. The actual identifier is calculated as follows:

node-ID + `COP_k_S_ID_RxPDO1`

(5) Finally the client application deregisters from the DLL again with `COP_ReleaseBoard()`.
Attention: this must only be done if the application is closed and not after each access! It completely releases the CAN-board and removes the CANopen Master Firmware from the microcontroller.

You now have the necessary equipment to determine all node parameters of a CANopen-network through SDO read accesses, as also implemented in the additional demo application `NetManage.exe`

4.3 I/O-node with TPDO and RPDO Polling

The control of a simple I/O-node with a transmit-PDO (**TPDO**) for the outputs and a receive-PDO (**RPDO**) for the inputs introduces the use of PDOs in addition to the SDO-example.

After the last exercise you can now read out all necessary node parameters from the network. We use this to determine all parameters of the RPDO and TPDOs of the I/O-node.

(1) `COP_InitBoard()` as for exercise 4.2

(2) `COP_InitInterface()` as for exercise 4.2

(3) `COP_AddNode()` as for exercise 4.2

(4) With `COP_ReadSDO()` the following communication parameters are collected from the object dictionary (only necessary if the values are not already known):

PDO-parameters RPDO1 (outputs of the module)	**Index (hex)**	**Subindex (hex)**
CAN-ID	1400	1
Mode (synchronous, asynchronous)	1400	2
Number of application objects	1600	0
Application objects (Mapping)	1600	1..40

PDO-parameters TPDO1 (inputs of the module)	**Index (hex)**	**Subindex (hex)**
CAN-ID	1800	1
Mode (synchronous, asynchronous)	1800	2
Number of application objects	1A00	0
Application objects (Mapping)	1A00	1..40

(5) By calling `COP_CreatePDO()` with `type = COP_k_PDO_TYP_RX` the RPDO can now be adapted.
The CAN-ID (`CANid`), the transmission type (`mode`) and the data length in bytes (`length`) of the PDO are put together from the values determined. The data length is calculated from the sum of all object lengths of the application objects determined:

length = sum([1600sub01..40] & 0x000000ff) / 8

(6) By calling `COP_CreatePDO()` with `type = COP_k_PDO_TYP_TX` the TPDO is also updated.
The CAN-ID (`CANid`), the transmission type (`mode`) and the data length in bytes (`length`) of the PDO are put together from the values determined. The data length is calculated again from the sum of all object lengths of the application objects determined:
length = sum([1A00sub01..40] & 0x000000ff) / 8

(7) The node is now started with `COP_StartNode()`.
To be able to transmit or receive PDOs, the relevant node must be set to **Operational** mode.

(8) The I/O-outputs can now be set by the application with `COP_WritePDO()`.

(9) To query the I/O-inputs determined by the module, `COP_ReadPDO()` must be called cyclically. This is normally done in a loop until all PDOs are read out of the RPDO-Queue.

(10) To close the program this time, the node (or the whole network) is stopped with the NMT-command `COP_StopNode()`. A restart is facilitated if the node is simply put on standby with `COP_EnterPreOperational()`.

(11) Then the application must call `COP_ReleaseBoard()` again as last command.

4.4 I/O-node with TPDO and RPDO Callback

The signalling of the receipt of RPDOs, emergency objects, status messages and Sync-objects replaces the Polling of the Master Firmware.

This exercise is an addition to the previous exercise with Polling. Alternatively to the declaration of Callback-functions, Windows messages can also be defined by means of `COP_DefineMsgRPDO` `COP_DefineMsgEvent` `COP_DefineMsgEmergency` `COP_DefineMsgSync`, which are posted for signalling to a window of the Client application.

(1) to (6) as previous example.

(7) So that the application can be called back by the DLL, a function of type `COP_t_EventCallback` must be implemented.

(8) The address of this Callback function is declared with `COP_DefineCallbacks()` as parameter `fp_rx_pdo` with the CANopen Master API.

(9) With `COP_StartNode()` the node (the network) can be started again.

(10) If the CANopen-node transmits a PDO, this is entered in the RPDO-Queue by the Master Firmware and the `fp_rx_pdo` Callback function is called with the queue number as parameter.

(11) In the Callback the application must read out the RPDO-Queue with `COP_CreatePDO()` until it is empty.
It is to be noted here that the application program is in the Thread-context of the Master API DLL and thus protected elements (members) of the application are not accessible or data pointers may be incorrect. After exiting the Callback function, the callback thread of the calling Master API DLL is returned to.

(12) Additional transmission of the PDO can be triggered with `COP_RequestPDO()`.

(13) If the application is to be closed, follow the procedure in the previous examples.

4.5 Altering the PDO-mode

Here an example is given of the (re-)configuration of a node, so that PDOs are now transmitted synchronously (only after receipt of the synchronization object) instead of asynchronously (transmitted by node on own initiative).

A separate subindex exists in the communication record of the PDOs for the mode (**Transmission type**) of a process data object. In example 4.3 it has already been read out (RPDO=[1400sub02]; TPDO=[1800sub02]). This entry is now to be written on. This allows the mode to be altered.

This example builds on the previous examples.

(1) to (4) as in example 4.3

(5) Now the TPDO of the node is set to synchronous transmission. For this a 1 is written on [1800sub02] by means of `COP_WriteSDO()`.

(6) Before the synchronization object is transmitted by the Master Firmware, it can be configured with the function `COP_DefSyncObj()`. Here it is possible to set the cycle time (`sync_period`).

(7) The cyclic transmission of the Sync-object by the Master Firmware is now started with `COP_EnableSync()` and can be stopped again with `COP_DisableSync()`.

(8) As only the mode of the transmit-PDO has been altered, it is possible to proceed as in example 4.3 under (5) to (11).

However it is necessary here to briefly discuss the differences between synchronous and asynchronous transmission. For further information on this, see Appendix F - CANopen-specific aspects.

- Asynchronous transmission:
  A node transmits a PDO when for example a change in value has occured or it is requested to do so by a PDO-request.
- Synchronous transmission:
  As soon as the network-Master transmits a Sync-object, the PDO is sent by the node. The CANopen-Master stores the PDO.
  When the next Sync-object is received, all previously received synchronous PDOs of the last Sync-cycle are passed on to the Client application. The Receive-PDOs therefore always run behind a Sync-period (as required in the specification).

5 Individual functions

This section contains the complete function reference of the CANopen Master API.

The Function prototypes are found in the language-dependent header file `cop.h`, `cop.vb`, `cop.cs` or `cop.pas`

5.1 Basic API-functions

The basic API functions are used for the initialization and parameterization of the API, selection of the CAN-board and for communication with the Master Firmware on the CAN-board.

5.1.1 COP_InitBoard

Description: With `COP_InitBoard` an IXXAT CAN-board is allocated for use by the CANopen Master API. When first called, the CAN-board is reset, the Master Firmware loaded onto the board and started and the communication queues created.
With a subsequent call, additional CAN lines are activated. This function shall be called just once for every CAN line. Therefore, to use both CAN lines of a CAN board equipped with two CAN controllers, two calls are necessary, which only differ in the first parameter `pBoardhdl` and in the last parameter `lCANline`.

If the function was successful, a Boardhandle is returned that identifies the CAN board/line combination unmistakeably. This Boardhandle is transfered as first parameter with every function of the CANopen Master API.
As from Version 6 CANopen Master API supports all VCI3 CAN boards – even those for which there is no specific firmware (see chapter 2.2) available. In such cases, the so called generic VCI3 firmware is being utilised, which is a windows DLL included in the scope of delivery. It supports up to 4 CAN controllers per board. Master API DLL is capable of handling a maximum of 12 CAN boards simultaneously.

Prototype:

```
short COP_InitBoard( COP_t_HANDLE* pBoardhdl,
                     GUID*         pBoardtype,
                     GUID*         pBoardID,
                     long          lCANline );
```

Parameters:

Parameter	Dir.	Explanation
pBoardhdl	(out)	Identifies this CAN board/line combination with all subsequent function calls.
pBoardtype	(in/out)	Type of the CAN-board according to header file vcguid resp. xatbrds See section 2.2 for a list of those CAN-boards, that are supported by means of a custom firmware: GUID_IPCIXC16PCI_DEVICE, GUID_IPCIXC16PCIE_DEVICE, GUID_IPCI320PCI_DEVICE, GUID_IPCI320104_DEVICE, GUID_TINCANV4_DEVICE, GUID_USB2CANCOMPACT_DEVICE, GUID_USB2CANII_DEVICE, GUID_TINCAN161_DEVICE, GUID_CANATNET2_DEVICE, GUID_CANIB200_PCIE_DEVICE. Also, there are two special values defined in the main header cop: COP_DEFAULTBOARD means that the one board defined in the VCI2 IXXAT Control Panel Applet, the so-called standard CAN-board (marked there in blue) should be used. With VCI3, simply the only one installed CAN board will be used. This is the typical application with exactly one CAN-board in the computer. COP_BOARDDIALOG means that a board selection dialog is to be displayed, from which the user himself then selects the CAN-board to be used. Value is always returned at COP_DEFAULTBOARD and COP_BOARDDIALOG and can be saved for example in the persistent configuration data of the Client application.

pBoardID	(in/out)	Unique identifier of a CAN-board. Is used together with pBoardtype in order to unmistakeably identify a board locally. If only one board of the respective type is present, enter the value COP_1stBOARD here. Several installed CAN-boards of the same type can be detected and differentiated by the Client application giving consecutive pBoardID values in each case for the same pBoardtype (beginning with 0). In this way, the n-th board of this type is taken. For this purpose, there are respective tokens COP_1stBOARD, COP_2ndBOARD, COP_3rdBOARD etc already #defined in the header. Value is always returned and can be saved for example in the persistent configuration data of the Client application.
lCANline	(in)	Selection of the CAN line to be used. Several default values are defined in the main header cop for this: COP_FIRSTLINE — First CAN line. This is the default value. COP_SECONDLINE — Second CAN line (if it exists). COP_THIRDLINE — Third CAN line (if it exists). COP_FOURTHLINE — Fourth CAN line (if it exists). COP_SINGLELINE — Single line firmware. In the case of applications where maximum performance is required using only the first CAN line. In this way, a specially optimized single line firmware is loaded onto the CAN board, that due to axed CAN differentiation can work somewhat faster.

Return values:

Return value	Description
BER_k_OK	Success
BER_k_ERR	General error, not further specified
BER_k_BOARD_ALREADY_USED	Required CAN-board already being used by CANopen Master API
BER_k_ALL_BOARDS_USED	Master API has reached maximum capacity of 4 simultaneously operable CAN-boards
BER_k_CANNOT_SEARCH_BOARD	No board selected, as the user has cancelled the IXXAT hardware-selection dialog with “Cancel”-button
BER_k_BOARD_NOT_FOUND	Specified board type and key do not match any available CAN-board

BER_k_BOARD_NOT_SUPP	Required CAN-board is not supported by CANopen Master API due to unsuitable Microcontroller or memory extension
BER_k_WRONG_FW	The version number supplied by the firmware is unsuitable. Indicates faulty communication between PC and μC
BER_k_USED_FROM_OTHER_PROCESS	Required CAN-board is already occupied by another CAN-application
BER_k_PC_MC_COMM_ERR	No communication could be set up with the CAN-board.
BER_k_BOARD_DLD_ERR	An error has occurred during firmware download. Most probable reason: The I/O-address range is being used by another hardware or driver This error also indicates that the generic VCI3 firmware library is missing or could not be loaded.
BER_k_NO_SUCH_CANLINE	Required CAN line does not exist or is not supported by the firmware
BER_k_CANLINE_USED	Required CAN line is already in use
BER_k_VCI_INST_ERR	Basic driver VCI not available or defective
BER_k_BOARD_ERR	Incorrect or unknown board type
BER_k_CCI_INST_ERR	CCI installation error (internal)
BER_k_SDO_INST_ERR	Internal error when instancing or configuring the SDO handler (internal)

5.1.2 COP_ReleaseBoard

Description: With `COP_ReleaseBoard`, an IXXAT CAN board/line combination used by the CANopen Master API is cancelled and the board released where appropriate.
This function must always be called exactly once for each board/line combinationused by `COP_InitBoard()` in order to reset the relevant CAN controller. Only when the last CAN line of a board has been released in this way is the Master Firmware unloaded and the board released for other applications.
If on the other hand another CAN line is in operation, another one can be activated at any time with `COP_InitBoard()`.

Prototype: `void COP_ReleaseBoard( COP_t_HANDLE boardhdl )`

Parameters:

Parameter	Dir.	Explanation
`boardhdl`	(in)	Handle of the CAN-board/line combination

Return values:

none

5.1.3 COP_GetBoardInfo

Description: With COP_GetBoardInfo information on the hardware properties of the CAN-board used and the version numbers of the software components are queried.

Prototype:

```
short COP_GetBoardInfo( COP_t_HANDLE     boardhdl,
                        COP_BOARD_INFO* sp_info );
```

Parameters:

Parameter	Dir.	Explanation
Boardhdl	(in)	Handle of the CAN-board/line combination
sp_info	(out)	Pointer to a buffer provided by the Client-application of data type COP_BOARD_INFO, which records the information

COP_BOARD_INFO Alignment: 1 byte

Field	Type	Meaning
hw_version	WORD	Revision number of the CAN-board e.g. 0x0101-> V 1.01
fw_version	WORD	Version number of the Master Firmware e.g. 0x0640-> V 6.40
sw_version	WORD	Version number of the CANopen Master API e.g. 0x0600-> V 6.00
board_seg	DWORD	(I/O-address of the board used) Legacy element, it is no longer used with Master API 6, except for: If the generic VCI3 firmware is running, the value 0x100 is returned here.
irq_num	WORD	(Interrupt request line IRQ used by the board) Legacy element, it is no longer used with Master API 6
canlines	WORD	Number of supported CAN lines
serial_num[16]	char[]	Serial number of the CAN-board
str_hw_type[40]	char[]	Description of the card type

Return values:

Return value	Description
BER_k_OK	Success
BER_k_ERR	Handle invalid
COP_k_IV	NULL pointer as parameter

5.1.4 COP_InitInterface

Description: With `COP_InitInterface` the firmware is parameterized on the CAN-board.
Here the baudrate of the network and the node monitoring mechanism to be used by the CANopen Master Firmware is defined.
The call of `COP_InitInterface` together with `COP_InitBoard` forms a logical unit and the successful execution of these two initializers is a condition for all further functions of the other function categories of the CANopen Master API.
Comments on the value range and the resolution of `hbTime` can be found in Appendix H - Timer resolutions and value ranges.

Contrary to most of the other API functions, this one must not be called again during operation, because it performs basic initialisations, amongst other things, of the internal firmware data.
Only the heartbeat time of the firmware might be changed retroactively by calling COP_ChangeNodeParameter().

Prototype:

```
short COP_InitInterface( COP_t_HANDLE boardhdl,
                         BYTE         baudtable,
                         BYTE         baudrate,
                         BYTE         node_no,
                         WORD         hbTime,
                         DWORD        AddFeatures );
```

Parameters:

Parameter	Dir.	Explanation
`boardhdl`	(in)	Handle of the CAN-board/line combination
`baudtable`	(in)	Number of the baudrate table to be used. There are two different tables: `COP_k_BAUD_CIA` Table with the baudrates specified by CiA in CiA-301. Standard table. `COP_k_BAUD_USER` Table with userdefined bittiming values. These must have been set before by a further API function.

baudrate	(in)	The predefined baudrates of both tables. The following values are permissible: COP_k_10_KB COP_k_20_KB COP_k_50_KB COP_k_100_KB COP_k_125_KB COP_k_250_KB COP_k_500_KB COP_k_800_KB COP_k_1000_KB
node_no	(in)	Node-ID of the CANopen Master Firmware between 1 and 127. A node-ID that diverges from the standard value 0 is only necessary if the CANopen network is operated with the Heartbeat node monitoring mechanism. The health of CANopen Master Firmware can thus be monitored by other subscribers.
hbTime	(in)	Heartbeat-interval of the CANopen Master Firmware in milliseconds; 0 for deactivated.
AddFeatures	(in)	Activation of additional functionality. There are three different predefined values: COP_k_NO_FEATURES No additional functionality. Default value. COP_k_FEATURE_FLYING_MASTER Activation of the Flying Master functionality in accordance with CiA-302. Is not supported by all board types (see Appendix B - Performance characteristics). COP_k_FEATURE_LOWSPEED Activation of Low Speed bus coupling instead of the default HighSpeed coupling. Note that the baudrate is limited to 125kB (COP_k_125_KB) with LowSpeed.

Return values:

Return value	Description
COP_k_OK	Success
COP_k_CAL_ERR	General error of the Master Firmware
COP_k_IV	Invalid parameter value
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Operation could not be entered in the transmit-command queue
BER_k_TIMEOUT	No response from Master Firmware
BER_k_CANLINE_USED	CAN line already initialised
COP_k_NO_FLY_MASTER_PRESENT	Flying Master functionality not supported
COP_k_NO_LOWSPEED	LowSpeed bus coupling not present or not supported

5.1.5 COP_DefineCallbacks

Description: With `COP_DefineCallbacks,` functions from the Client-application of type `COP_t_EventCallback` are declared to the CANopen Master API, which are then called by the API in the case of a new queue object.
There are four separate queues for the receipt of CANopen communication objects. These are the PDO-receive queue for process data objects, the EMCY-queue for emergency objects, the Event-queue for network- or firmware-events and the Sync-queue for Sync-objects. A separate callback function can be defined for each of these four queues or `NULL` if not required. Within the PDO callback function, `COP_ReadPDO()` should be called until the PDO-receive queue is empty.
In the Emergency Callback function, `COP_GetEmergencyObj()` is called until the EMCY-queue is emptied.
The same applies to the Event and Sync-callback function. Here the functions `COP_GetEvent()` or `COP_CheckSync()` resp. are used to read out the queues.
It is possible to call this API function several times when the program is running, for example to "deregister" callback functions temporarily, to "re-register" or to define additional callback functions.
As the `boardhdl` identifies a specific CAN board/line combination, different callback functions can also be defined for the various CAN lines.

The Sync-queue is filled internally with a message for each Sync-object after the function () is called. For this reason the Sync-Queue must regularly be read until empty with the function COP_CheckSync(). If this is not done, increased CPU-load is to be expected after a certain time due to internal queue overruns.

Prototype:

```
short COP_DefineCallbacks(
                        COP_t_HANDLE        boardhdl,
                        COP_t_EventCallback fp_rx_pdo,
                        COP_t_EventCallback fp_emergency,
                        COP_t_EventCallback fp_net_event,
                        COP_t_EventCallback fp_sync );
```

Parameters:

Parameter	Dir.	Explanation
Boardhdl	(in)	Handle of the CAN-board/line combination
fp_rx_pdo	(in)	This function is called when a process data object (PDO) has been received. See also the description of COP_ReadPDO()
fp_emergency	(in)	This function is called when an emergency object (EMCY) has been received. See also the description of COP_GetEmergencyObj()
fp_net_event	(in)	This function is called when a network event has occured. See also the description of COP_GetEvent()
fp_sync	(in)	This function is called when a (self-transmitted) Sync-Object has been received. See also the description of COP_CheckSync()

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_BADCALLBACK_PTR	An invalid function pointer is given
COP_k_OK	Success

5.1.6 COP_t_EventCallback

Description: COP_t_EventCallback is a function prototype for functions within the Client-application, which are registered with the CANopen Master API with COP_DefineCallbacks() and called for signalling of a new object in a receive-data queue.

Prototype:

```
typedef void (CALLBACK* COP_t_EventCallback)(
                                COP_t_HANDLE boardhdl,
                                UINT8        que_num,
                                UINT8        canline );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
que_num	(in)	Contains the queue ID. The following values are possible: COP_M2P_QUEUE_PDO0 bzw. COP_M2P_QUEUE_PDO1 COP_M2P_QUEUE_EMERGENCY0 bzw. COP_M2P_QUEUE_EMERGENCY1 COP_M2P_QUEUE_EVENT0 bzw. COP_M2P_QUEUE_EVENT1 COP_M2P_QUEUE_SYNC0 bzw. COP_M2P_QUEUE_SYNC1 The last digit of the constants stands in each case for the CAN line
canline	(in)	CAN line, on which the new object was received

Return values:

none

5.1.7 COP_DefineMsgRPDO
COP_DefineMsgEvent
COP_DefineMsgEmergency
COP_DefineMsgSync

Description: With these functions Windows-messages are defined that are posted to a window of the Client application in the case of a new queue object.
The `wParam` of the message contains the Boardhandle, the `lParam` contains the queue number similar to parameter `que_num` of function prototype `COP_t_EventCallback`.
There are four separate queues for the receipt of CANopen communication objects. These are the PDO-receive queue for process data objects, the EMCY-Queue for emergency objects, the Event-Queue for network- or Firmware-events and the Sync-Queue for Sync-objects.
For each of these four queues with the correspondent function `COP_DefineMsgRPDO()`, `COP_DefineMsgEvent()`, `COP_DefineMsgEmergency()`, `COP_DefineMsgSync()` a separate Windows-message, thread message or both can be defined. If observation of a queue is undesired, `0` can also be entered for the corresponding function parameter.
Within the PDO Message Handler, `COP_ReadPDO()` should be called until the PDO-receive queue is empty.
In the Emergency Message Handler, `COP_GetEmergencyObj()` is to be called until the EMCY-Queue is empty.
The same applies to the Event and Sync-Message Handler. Here the functions `COP_GetEvent()` or `COP_CheckSync()` resp. are used to read out the queues.
It is possible to call this API function several times when the program is running, for example to "deregister" callback functions temporarily, to "re-register" or to define additional callback functions.
As the `boardhdl` identifies a specific CAN board/line combination, different callback functions can also be defined for the various CAN lines.

After calling the function COP_EnableSync(), the Sync-Queue is filled with a message internally for each Sync-object. For this reason the Sync-Queue must regularly be read until empty with the function COP_CheckSync().

If this is not done, increased CPU-load is to be expected after a certain time due to internal queue overruns.

Prototype:

```
short COP_DefineMsgCCCC( COP_t_HANDLE boardhdl,
                         HWND         hWnd,
                         DWORD        idThread,
                         UINT         Msg );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
hWnd	(in)	Handle of the window to which given Msg message is to be posted. A value of 0 deactivates the message posting.
idThread	(in)	Identifier of the thread to which given Msg message is to be posted. A value of 0 deactivates the message posting.
Msg	(in)	This message is posted when an object of the respective queue has been received.

Return values:

Return value	Description
BER_k_ERR	Handle invalid
COP_k_IV	An invalid window handle was given
COP_k_OK	Success

5.1.8 COP_GetThreadIds

Description: The thread identifiers of the internal CANopen Master API DLL poll threads for the recieve queues can be read out with `COP_GetThreadIds`.
A separate poll thread is started in the Master API DLL for each of the four queues for the reception of CANopen communication objects if a corresponding callback function or message has been defined for the client application. These poll threads are instanced and destroyed as required when the API functions `COP_DefineCallbacks()` and
`COP_DefineMsgRPDO`
`COP_DefineMsgEvent`
`COP_DefineMsgEmergency`
`COP_DefineMsgSync` are called.
The same poll thread is used for queues of the same name of all CAN lines of one board.

This `COP_GetThreadIds()` function enables the client application immediate access to the relevant poll thread in the Master API DLL via the Windows function `OpenThread()`, for example in order to change its priority or to stop it. As these interventions possibly have serious consequences for the stability of the Master API DLL or of the client application, they are generally not recommended.

Prototype:

```
short COP_GetThreadIds(
                        COP_t_HANDLE boardhdl,
                        PUINT        pPdoThreadId,
                        PUINT        pEmcyThreadId,
                        PUINT        pEventThreadId,
                        PUINT        pSyncThreadId );
```

Parameters:

Parameter	Dir.	Explanation
`Boardhdl`	(in)	Handle of the CAN board/line combination
`pPdoThreadId`	(out)	Identifier of the poll thread for the PDO receive queues
`pEmcyThreadId`	(out)	Identifier of the poll thread for the EMCY queues
`pEventThreadId`	(out)	Identifier of the poll thread for the event queues
`pSyncThreadId`	(out)	Identifier of the poll thread for the sync queues

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_CCI_INST_ERR	Initialization of the board incomplete or defective
BER_k_OK	Success

5.1.9 COP_Reset_DLL

Description: With `COP_ResetDLL` the CANopen Master DLL is re-initialized to be able to register the board again in the event of a program abort (without release of the board) in an interpreter debugger such as Visual Basic.

All registered CAN-boards and all CAN lines will be deregistered.

Prototype: `void COP_Reset_DLL();`

Parameters:

none

Return values:

none

5.1.10 COP_SendMsg

Description: With COP_SendMsg an entry of type COP_t_Message is written to the transmit-command queue.
A list of the record fields and the supported command-Opcodes is given in Appendix D - Data structures of the command queues.

Prototype:

```
short COP_SendMsg( COP_t_HANDLE   boardhdl,
                   COP_t_Message* sp_message );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
sp_message	(in)	Pointer to the operation record for the Master Firmware

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Operation could not be entered in the transmit-command queue
COP_k_OK	Success

5.1.11 COP_GetMsg

Description: With `COP_GetMsg` an entry of type `COP_t_Message` is read out of the receive-command queue.
A list of the record fields and the supported command-Opcodes is given in Appendix D - Data structures of the command queues.

Prototype:

```
short COP_GetMsg( COP_t_HANDLE   boardhdl,
                  COP_t_Message* sp_message );
```

Parameters:

Parameter	Dir.	Explanation
`Boardhdl`	(in)	Handle of the CAN-board/line combination
`sp_message`	(out)	Pointer to the Confirmation record for the response of the Master Firmware

Return values:

Return value	Description
`BER_k_ERR`	Handle invalid
`BER_k_TIMEOUT`	Delay time expired. No entry found in the Receive-command queue.
`BER_k_PC_MC_COMM_ERR`	Internal error during access of the receive command queue.
`BER_k_DATA_CORRUPT`	Wrong sequence number in firmware response. Failure of transmission path (USB, ethernet, …) from windows-DLL to device firmware.
`COP_k_OK`	Success

5.1.12 COP_SetCommTimeOut

Description: With `COP_SetCommTimeOut` the delay time is defined that determines how long to wait for an acknowledgement of the Master Firmware.
With almost every function of the CANopen Master API, an operation record is compiled within the DLL for the Master Firmware, transfered to it by means of `COP_SendMsg()` and in `COP_GetMsg()` processing or confirmation by the Master Firmware is awaited as described in section 3.2 Internal use of the command queues. If the set delay time is exceeded, the relevant function returns with the return value `BER_k_TIMEOUT`.
The standard value for the delay time is 5 seconds.

This communication delay time is internally coupled to the SDO delay time, as the SDO delay time is subordinate to the communication delay time. When the communication delay time is set to a value less than the SDO delay time, the SDO delay time is therefore also automatically reduced.

Prototype:

```
short COP_SetCommTimeOut( COP_t_HANDLE boardhdl,
                          WORD         w_timeout );
```

Parameters:

Parameter	Dir.	Explanation
`boardhdl`	(in)	Handle of the CAN-board/line combination
`w_timeout`	(in)	New value for the delay time in milliseconds. The value range is 55 <= w_timeout <= 65535. Smaller values are rounded up internally.

Return values:

Return value	Description
`BER_k_ERR`	Handle invalid
`BER_k_OK`	Success

5.1.13 COP_GetStatus

Description: With `COP_GetStatus` the status of the Master Firmware and the status of the Data Link Layer on the CAN-board are queried.
A change of the status of the data link layer is also signaled via event-queue and can be read out by calling `COP_GetEvent()`.

Prototype:

```
short COP_GetStatus( COP_t_HANDLE boardhdl,
                     BYTE*        state_master,
                     BYTE*        state_err_dll );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
state_master	(out)	Status of the CANopen Master Firmware. The following values are possible: COP_k_INIT Master is parameterized and ready for use (function COP_InitInterface() has already been called) COP_k_NOT_INIT Master has not yet been parameterized by function COP_InitInterface()
state_err_dll	(out)	Status of the Data Link Layer of the Master Firmware. The following values are possible: COP_k_DLL_NOERR No error COP_k_DLL_RXOVR Receive queue overrun COP_k_DLL_TXOVR Transmit queue overrun COP_k_DLL_COVR CAN-Controller: overrun COP_k_DLL_BOFF CAN-Controller: In Bus-Off state COP_k_DLL_ESET CAN-Controller: Error-Statusbit set COP_k_DLL_ERESET CAN-Controller: Error-Statusbit reset

Return values:

Return value	Description
BER_k_OK	Success
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Operation could not be entered in the transmit-command queue
BER_k_TIMEOUT	No response from Master Firmware

5.1.14 COP_TestCommand

Description: With `COP_TestCommand` it is possible to determine whether the firmware has been correctly loaded onto the CAN-board and started.
For this a test string is requested and checked. At the same time it is also checked whether the communication between CANopen Master API DLL and the Master Firmware is working correctly via both command queues.

Prototype:
```
short COP_TestCommand( COP_t_HANDLE boardhdl );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Operation could not be entered in the transmit-command queue; transmit-command queue not available
BER_k_TIMEOUT	No response from Master Firmware; firmware not started or receive-command queue not available
BER_k_DATA_CORRUPT	Corrupt data received, Communication path (USB, Ethernet) disturbed
COP_k_OK	Firmware started, communication working.

5.2 Functions for the network management

The functions for the network management are used to set up the network and control the individual CANopen nodes.

5.2.1 COP_AddNode

Description: With `COP_AddNode` a new node is registered with the Master Firmware and thus added to the network management.
With this process internal management structures and status variables are initialized, so that the node registration is the condition for any communication with the corresponding CANopen device. In addition, the first 4 receive- and transmit-PDOs according to **Predefined Connection Set** are established, so `COP_CreatePDO()` does not have to be called for these 8 PDOs.
The parameter `NgOrHb` states whether the node monitoring is carried out via Node-Guarding or by means of Heartbeat-message. Mixed mode of the two node monitoring mechanisms is permitted.
In the case of Heartbeat it's up to the Client Application to configure the corresponding Object Dictionary entry [1017.0] (Producer Heartbeat Time) of the node appropriately, so that the node actually generates heartbeat messages.
The node monitoring begins when the function `COP_StartNode()` or `COP_EnterPreOperational()` is called (see also appendix F - Node guarding and node states).
The current state of a network participant can be queried by `COP_GetNodeState()`.
Information on the value range and the resolution of the `GuardHeartbeatTime` is given in Appendix H - Timer resolutions and value ranges.

Depending on the CAN-board used, the number of nodes that can be managed simultaneously varies. USB-to-CAN compact as well as all 320-based boards are not able to support 127 nodes simultaneously. Refer to Appendix B - Performance characteristics for the exact values.

Prototype:

```
short COP_AddNode( COP_t_HANDLE boardhdl,
                   BYTE         node_no,
                   BYTE         NgOrHb,
                   WORD         GuardHeartbeatTime,
                   BYTE         lifetimefactor );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
node_no	(in)	Node-ID of the external CANopen device (between 1 and 127)
NgOrHb	(in)	Definition of the monitoring mechanism applicable for the node. The following values are permitted: COP_k_NODE_GUARDING Node is to be guarded, i.e. cyclic request of a predefined toggling CAN-telegram by the Master COP_k_HEARTBEAT Node sends Heartbeat message cyclically, i.e. node sends a predefined CAN-telegram on own initiative
GuardHeartbeatTime	(in)	Guardtime in milliseconds. With COP_k_NODE_GUARDING this parameter states the so-called Guardtime, i.e. the time period between two Guarding-request telegrams. 0 means that the node is not to be guarded at all. With COP_k_HEARTBEAT this parameter states the so-called Heartbeattime, i.e. the time period between two Heartbeat-messages.
lifetimefactor	(in)	Defines the number of unsuccessful attempts at a Guard-request by the Master to be permitted (between 1 and 255). If this number is exceeded without a response of the node being received, the firmware signals a COP_k_NMT_EVT in the Event-Queue. Only for COP_k_NODE_GUARDING.

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Operation could not be entered in the transmit-command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Success
COP_k_CAL_ERR	Required CAN resource not available
COP_k_IV	Invalid parameter value

5.2.2 COP_DeleteNode

Description: With `COP_DeleteNode` a node is removed from the internal node list of the CANopen-Master and therefore from the network-management.
The external node should first be stopped with NMT commands or reset, so that the network remains in a defined state.

Prototype:

```
short COP_DeleteNode( COP_t_HANDLE boardhdl,
                      BYTE         node_no );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
node_no	(in)	Node-ID of the external CANopen device (between 1 and 127)

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Operation could not be entered in the transmit-command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Success
COP_k_NOT_FOUND	No node registered with the stated node-ID
COP_k_IV	Invalid node-ID

5.2.3 COP_SearchNode

Description: COP_SearchNode checks whether a device is available in the network with the specified node-ID.
Before this function is called, the potential node must have been registered with COP_AddNode(). The check is carried out by attempting to access the mandatory object dictionary entry [1000] via the standard server-SDO of the node. The Timeout value used is 100 ms and is independent of the SDO Timeout.

Prototype:

```
short COP_SearchNode( COP_t_HANDLE boardhdl,
                      BYTE         node_no );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
node_no	(in)	Node-ID of the CANopen device to be searched for (between 1 and 127)

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Operation could not be entered in the transmit-command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Success
COP_k_NOT_FOUND	No node registered with the stated node-ID
COP_k_TIMEOUT	No (registered) node with the stated node-ID found in the network
COP_k_IV	Invalid node-ID

5.2.4 COP_GetNodeInfo

Description: COP_ChangeNodeParameter delivers the properties of a registered node.
They might be useful, for example, to find out the set node monitoring time. In order to change node settings call COP_ChangeNodeParameter() and COP_SetEmcyIdentifier() respectively.
Before this function is called, the potential node must have been registered with COP_AddNode().

Prototype:

```
short COP_GetNodeInfo(
                    COP_t_HANDLE boardhdl,
                    BYTE         node_no,
                    BYTE*        NgOrHb,
                    WORD*        GuardHeartbeatTime,
                    BYTE*        lifetimefactor,
                    WORD*        EmcyIdentifier );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
node_no	(in)	Node-ID of the external CANopen device (between 1 and 127)
NgOrHb	(out)	Set monitoring mechanism for the node. COP_k_NODE_GUARDING Node is to be guarded, i.e. cyclic request of a pre-defined toggling CAN-telegram by the Master COP_k_HEARTBEAT Node cyclically sends Heartbeat message, i.e. node sends a pre-defined CAN-telegram on own initiative
GuardHeartbeatTime	(out)	Set monitoring time in in milliseconds. With COP_k_NODE_GUARDING this parameter states the so-called Guardtime, i.e. the time period between two Guarding-request telegrams. 0 means that the node is not to be guarded at all. With COP_k_HEARTBEAT this parameter states the so-called Heartbeattime, i.e. the time period between two Heartbeat-messages.
lifetimefactor	(out)	Set number of unsuccessful attempts at a Guard-request by the Master to be permitted (between 1 and 255). If this number is exceeded without a response of the node being received, the firmware signals a COP_k_NMT_EVT in the Event-Queue. Only for COP_k_NODE_GUARDING.

EmcyIdentifier	(out)	Set CAN-identifier of the emergency object of the node. According to Predefined Connection Set its default value is 0x80+node_no. However, it might be changed by overwriting OD entry [1014] of the node, and then the Master firmware needs to follow by calling COP_SetEmcyIdentifier().

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Operation could not be entered in the transmit-command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Success
COP_k_NOT_FOUND	No node registered with the stated node-ID
COP_k_IV	Invalid parameter value

5.2.5 COP_ChangeNodeParameter

Description: COP_ChangeNodeParameter subsequently changes the properties of a node registered with COP_AddNode(). Information on the value range and the resolution of the GuardHeartbeatTime is given in appendix Appendix H - Timer resolutions and value ranges.

Prototype:

```
short COP_ChangeNodeParameter(
                    COP_t_HANDLE boardhdl,
                    BYTE         node_no,
                    BYTE         NgOrHb,
                    WORD         GuardHeartbeatTime,
                    BYTE         lifetimefactor );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
node_no	(in)	Node-ID of the CANopen device (between 1 and 127)
NgOrHb	(in)	Definition of the monitoring mechanism applicable for the node. The following values are permitted: COP_k_NODE_GUARDING Node is to be guarded, i.e. cyclic request of a pre-defined toggling CAN-telegram by the Master COP_k_HEARTBEAT Node cyclically sends Heartbeat message, i.e. node sends a pre-defined CAN-telegram on own initiative
GuardHeartbeatTime	(in)	Monitoring time in in milliseconds. With COP_k_NODE_GUARDING this parameter states the so-called Guardtime, i.e. the time period between two Guarding-request telegrams. 0 means that the node is not to be guarded at all. With COP_k_HEARTBEAT this parameter states the so-called Heartbeattime, i.e. the time period between two Heartbeat-messages.
lifetimefactor	(in)	Defines the number of unsuccessful attempts at a Guard-request by the Master to be permitted (between 1 and 255). If this number is exceeded without a response of the node being received, the firmware signals a COP_k_NMT_EVT in the Event-Queue. Only for COP_k_NODE_GUARDING.

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Operation could not be entered in the transmit-command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Success
COP_k_NOT_FOUND	No node registered with the stated node-ID
COP_k_IV	Invalid parameter value

5.2.6 COP_SetEmcyIdentifier

Description: With `COP_SetEmcyIdentifier` the CAN-identifier of the emergency object of a node can be adapted.
Each network participant transmits its emergency object on a reserved individual CAN-identifier which is calculated in a standardised way according to the node-ID. Since this emergency identifier can be reconfigured by writing to object dictionary entry [1014], CANopen Master API allows for adapting to such a reconfiguration by means of this function.

Calling this function is not required usually, because the firmware is able to receive all preset emergency-identifiers of all CANopen nodes by default.

Prototype:

```
short COP_SetEmcyIdentifier(
                        COP_t_HANDLE boardhdl,
                        BYTE         node_no,
                        WORD         EmcyIdentifier );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
node_no	(in)	Node-ID of the external CANopen device (between 1 and 127)
EmcyIdentifier	(in)	New CAN-identifier of the emergency object of the node. The standardised CAN-identifier is one of the highest priority of CANopen. It is calculated from the node-ID according to the following formula: EmcyIdentifier = 0x80 + node_no

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Operation could not be entered in the transmit-command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Success
COP_k_NOT_FOUND	No node registered with the stated node-ID
COP_k_IV	Invalid parameter value
COP_k_CAL_ERR	Given CAN-identifier already in use

5.2.7 COP_ConfigFlyMaster

Description: With `COP_ConfigFlyMaster`, the Flying Master functionality of the Master Firmware is configured once in accordance with CiA-302. These are the contents of the object directory entry Flying Master Timing Parameters [1F90] and the object directory entry Consumer Heartbeat Time [1016] for monitoring of the active master after transferring network mastership.
The condition is that this additional functionality was activated on initialization of the firmware by means of `COP_InitInterface( AddFeatures = COP_k_FEATURE_FLYING_MASTER ).`
It is not possible to call this function more than once to change the set values later.
Additional functionality Flying Master is not available with all CAN boards (see Appendix B - Performance characteristics)

Prototype:

```
short COP_ConfigFlyMaster(
                COP_t_HANDLE boardhdl,
                WORD         wDetectionTimeout,
                WORD         wNegotiationDelay,
                WORD         wPriorityLevel,
                WORD         wPrioritytimeslot,
                WORD         wNodeTimeslot,
                WORD         wCycletimeCd,
                WORD         wCycletimeTimeoutHbeat );
```

Parameters:

Parameter	Dir.	Explanation
`Boardhdl`	(in)	Handle of the CAN-board/line combination
`wDetectionTimeout`	(in)	Delay time until detection of the active network master, corresponds to object directory entry [1F90sub1]
`wNegotiationDelay`	(in)	Delay until negotiation of the network mastership among the master candidates, corresponds to object directory entry [1F90sub2]
`wPriorityLevel`	(in)	Priority level of the CANopen Master API, corresponds to object directory entry [1F90sub3] Valid values are 0 (high), 1 and 2 (low)
`wPrioritytimeslot`	(in)	Object directory entry [1F90sub4]
`wNodeTimeslot`	(in)	Object directory entry [1F90sub5]
`wCycletimeCd`	(in)	Object directory entry [1F90sub6]
`wCycletimeTimeoutHbeat`	(in)	Monitoring of the active master after transfer of network mastership, corresponds to object directory entry [1016]

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Order could not be entered in transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Success
COP_k_IV	Unauthorized parameter value
COP_k_UNKNOWN	Flying Master functionality not supported OR Function already successfully called
COP_k_NO_FLY_MASTER_PRESENT	Flying Master functionality not activated

5.2.8 COP_StartFlyMaster

Description: `COP_StartFlyMaster` starts the Flying Master functionality of the Master Firmware. The CANopen Master Firmware then actively participates in the negotiation of the active master with other potential masters and attempts to gain network mastership based on its settings.
The condition is that this additional functionality was activated on initialisation of the firmware with `COP_InitInterface( AddFeatures = COP_k_FEATURE_FLYING_MASTER )` and the Flying Master was configured with `COP_ConfigFlyMaster()`.

Prototype:
```
short COP_StartFlyMaster( COP_t_HANDLE boardhdl );
```

Parameter:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Order could not be entered in transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Success
COP_k_IV	Unauthorized parameter value
COP_k_UNKNOWN	Function already successfully called
COP_k_NO_FLY_MASTER_PRESENT	Flying Master functionality not activated

5.2.9 COP_GetStatusFlyMasterNeg

Description: COP_GetStatusFlyMasterNeg returns the current status of the negotiation of network mastership with other potential masters.
The condition is that this additional functionality was activated on initialization of the firmware with COP_InitInterface(AddFeatures = COP_k_FEATURE_FLYING_MASTER) and the Flying Master was configured with COP_ConfigFlyMaster(). In addition it must have been started after configuration with COP_StartFlyMaster(). This function can be called regularly to check whether the CANopen Master Firmware again has or no longer has network mastership. In addition, a separate event type is defined for firmware events that can be read out with COP_GetEvent().

Prototype:

```
short COP_GetStatusFlyMasterNeg(
                        COP_t_HANDLE boardhdl,
                        BYTE*        status,
                        BYTE*        masterid,
                        BYTE*        masterprio );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
status	(out)	Status of negotiation. The following values are possible: COP_k_FLY_MASTER CANopen Master Firmware has gained network mastership and is the active NMT Master COP_k_FLY_NOT_MASTER CANopen Master Firmware lost the negotiation and is no longer the active NMT Master COP_k_FLY_WAIT_BUSCONNECTION CANopen Master Firmware is not on CAN COP_k_FLY_NEGOTIATION_RUNNING The negotiation is running and not yet concluded
masterid	(out)	Node-ID of the active NMT Master if not CANopen Master Firmware itself
masterprio	(out)	Priority class of the active NMT Master if not CANopen Master Firmware itself

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Order could not be entered in transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Success
COP_k_IV	Unauthorized parameter value
COP_k_UNKNOWN	Flying Master functionality not supported
COP_k_NO_FLY_MASTER_PRESENT	Flying Master functionality not activated

5.2.10 COP_StartNode

Description: COP_StartNode transfers a single node or the whole network by transmitting an NMT-command into OPERATIONAL state.

Prototype:

```
short COP_StartNode( COP_t_HANDLE boardhdl,
                     BYTE         node_no );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
node_no	(in)	Node-ID of the node to be controlled (between 1 and 127) or 0 for all nodes.

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Operation could not be entered in the transmit-command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Success
COP_k_NOT_FOUND	No node registered with the stated node-ID
COP_k_IV	Invalid node-ID

5.2.11 COP_StopNode

Description: COP_StopNode transfers a single node or the whole network by transmitting an NMT-command into STOPPED state.

Prototype:

```
short COP_StopNode( COP_t_HANDLE boardhdl,
                    BYTE         node_no );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
node_no	(in)	Node-ID of the node to be controlled (between 1 and 127) or 0 for all nodes.

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Operation could not be entered in the transmit-command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Success
COP_k_NOT_FOUND	No node registered with the stated node-ID
COP_k_IV	Invalid node-ID

5.2.12 COP_ResetComm

Description: COP_ResetComm resets the values of the communication profile of a single node or of the whole network by transmitting an NMT-command.

Prototype:

```
short COP_ResetComm( COP_t_HANDLE boardhdl,
                     BYTE         node_no );
```

Parameters:

Parameter	Dir.	Explanation
Boardhdl	(in)	Handle of the CAN-board/line combination
node_no	(in)	Node-ID of the node to be controlled (between 1 and 127) or **0** for all nodes.

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Operation could not be entered in the transmit-command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Success
COP_k_NOT_FOUND	No node registered with the stated node-ID
COP_k_IV	Invalid node-ID

5.2.13 COP_ResetNode

Description: COP_ResetNode resets the application and the values of the communication profile of a single node or of the whole network by transmitting an NMT-command.

Prototype:

```
short COP_ResetNode( COP_t_HANDLE boardhdl,
                     BYTE         node_no );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
node_no	(in)	Node-ID of the node to be controlled (between 1 and 127) or 0 for all nodes.

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Operation could not be entered in the transmit-command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Success
COP_k_NOT_FOUND	No node registered with the stated node-ID
COP_k_IV	Invalid node-ID

5.2.14 COP_EnterPreOperational

Description: `COP_EnterPreOperational` transfers a single node or the whole network by transmitting an NMT-command into PRE-OPERATIONAL state.

Prototype:

```
short COP_EnterPreOperational(
                                    COP_t_HANDLE boardhdl,
                                    BYTE         node_no );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
node_no	(in)	Node-ID of the node to be controlled (between 1 and 127) or **0** for all nodes.

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Operation could not be entered in the transmit-command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Success
COP_k_NOT_FOUND	No node registered with the stated node-ID
COP_k_IV	Invalid node-ID

5.2.15 COP_GetNodeState

Description: COP_GetNodeState retrieves the current NMT state of a single CANopen node. It is being deduced from monitoring of communication with the slave, mainly by its heartbeat resp guarding responses, but also by its bootup message.

Prototype:

```
short COP_GetNodeState( COP_t_HANDLE boardhdl,
                        BYTE         node_no,
                        WORD*        node_state );
```

Parameters:

Parameter	Dir.	Erklärung
boardhdl	(in)	Handle of the CAN-board/line combination
node_no	(in)	Node-ID of the desired node (between 1 and 127)
node_state	(out)	Current state of the network participant The following values are possible: COP_k_NS_STOPPED Node is in STOPPED state COP_k_NS_OPERATIONAL Node is in OPERATIONAL state COP_k_NS_PREOPERATIONAL Node is in PRE-OPERATIONAL state COP_k_NS_UNKNOWN Node state is unknown or the node is not registed with Master Firmware. This default value will be returned especially as the NMT state could not be deduced (yet) due to e.g. lack of node monitoring or turning off of heartbeat. COP_k_NS_DISCONNECTED A monitoring error with the node has been encountered: Either heartbeat/guarding failed or there was an incomprehensible change in reported state by the Node itself. As part of the error handling an NMT command (COP_StartNode .. COP_EnterPreOperational) is recommended to get the node back on track.

Return values:

Rückgabewert	Beschreibung
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Operation could not be entered in the transmit-command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Success
COP_k_NOT_FOUND	No node registered with the stated node-ID
COP_k_IV	Invalid node-ID

5.3 CANopen object management

The functions for the CANopen-object management are used for creating and parameterizing CANopen communication objects.

5.3.1 COP_CreatePDO

Description: With `COP_CreatePDO` a process data object is created in the internal Master Firmware object management. The function call does not affect any external slave device. Hence this local PDO is just mirroring an existing PDO on a physical node.
By stating the node-ID the PDO is allocated to a node. If the value **0** is given as node-ID, the PDO is not directly allocated to a node and can be allocated to several nodes by appropriate identifier allocation to the node (via SDO).
It is also possible to call this function more than once for the same PDO to alter the properties of the PDO subsequently when the network has already been started.

The first 4 Transmit- and Receive-PDOs according to Predefined Connection Set do not need to be created explicitly. They are already established with data length 8 and transmission type COP_k_PDO_MODE_ASYNC when calling COP_AddNode().

The number of PDOs that can be handled simultaneously varies according to the type of CAN board used. USB-to-CAN compact as well as all 320-based boards support only 12+12 PDOs simultaneously, all other boards more. Refer to Appendix B - Performance characteristics for exact values.

Prototype:

```
short COP_CreatePDO( COP_t_HANDLE boardhdl,
                     BYTE         node_no,
                     BYTE         pdo_no,
                     BYTE         type,
                     BYTE         mode,
                     BYTE         length,
                     WORD         CANid );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
node_no	(in)	Node-ID of the node allocated to the PDO (between 1 and 127) or 0 for no allocation.
pdo_no	(in)	Number of the PDO (between 1 and 16).
type	(in)	Transmission direction of the PDO from point of view of the Master. The following values are permitted: COP_k_PDO_TYP_RX for a receive-process-data-object COP_k_PDO_TYP_TX for a transmit-process-data-object
mode	(in)	Mode of the PDO in consonance with coding of subindex1 (Transmission Type) of the CANopen PDO communication parameters in the Object Dictionary. The following special values are defined: COP_k_PDO_MODE_SYNC for a synchronous process data object (corresponds to Transmission Type = 1) COP_k_PDO_MODE_ASYNC for an asynchronous, i.e. event-controlled process data object (corresponds to Transmission Type = 254)
length	(in)	Byte length of the PDO between 1 and 8.
CANid	(in)	Identifier of the CAN-object used by the PDO.

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Operation could not be entered in the transmit-command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Success
COP_k_NOT_FOUND	No node registered with the stated node-ID
COP_k_CAL_ERR	Given CAN-identifier is already in use at receiving side
COP_k_IV	Invalid parameter value

5.3.2 COP_DeletePDO

Description: With COP_DeletePDO a process data object is removed from the internal Master Firmware object management and its resources (the CAN-identifier in particular) is being released.

Prototype:

```
short COP_DeletePDO( COP_t_HANDLE boardhdl,
                     BYTE         node_no,
                     BYTE         pdo_no,
                     BYTE         type );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
node_no	(in)	Node-ID of the node allocated to the PDO (between 1 and 127) or 0 for no allocation.
pdo_no	(in)	Number of the PDO (between 1 and 16).
type	(in)	Transmission direction of the PDO from point of view of the Master. The following values are permitted: COP_k_PDO_TYP_RX for a receive-process-data-object COP_k_PDO_TYP_TX for a transmit-process-data-object

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Operation could not be entered in the transmit-command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Success
COP_k_NOT_FOUND	No node registered with the stated node-ID
COP_k_IV	Invalid parameter value (pdo_no)

5.3.3 COP_GetPDOInfo

Description: COP_GetPDO delivers the properties of a process data object. In order to change these properties call function COP_CreatePDO().

Prototype:

```
short COP_GetPDOInfo( COP_t_HANDLE boardhdl,
                      BYTE         node_no,
                      BYTE         pdo_no,
                      BYTE         type,
                      BYTE*        mode,
                      BYTE*        length,
                      WORD*        CANid );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
node_no	(in)	Node-ID of the node allocated to the PDO (between 1 and 127) or 0 for no allocation.
pdo_no	(in)	Number of the PDO (between 1 and 16).
type	(in)	Transmission direction of the PDO from point of view of the Master. The following values are permitted: COP_k_PDO_TYP_RX for a receive-process-data-object COP_k_PDO_TYP_TX for a transmit-process-data-object
mode	(out)	Set mode of the PDO in consonance with coding of subindex1 (Transmission Type) of the CANopen PDO communication parameters in the Object Dictionary.
length	(out)	Set byte length of the PDO between 1 and 8.
CANid	(out)	Set identifier of the CAN-object used by the PDO.

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Operation could not be entered in the transmit-command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Success
COP_k_NOT_FOUND	No node registered with the stated node-ID
COP_k_IV	Invalid parameter value

5.3.4 COP_CreateSDO

Description: With COP_CreateSDO an *additional* client service data object is created in the internal Master Firmware object management (the default-SDO already exists for each with COP_AddNode() registered node). Provided, however, an appropriate ServerSDO object is existing on the registered node. Configuration of the same with the respective Object Dictionary entry [1201]..[127F] is the responsibility of the client application, that is the user.
The CANopen Master API is always the SDO-client, the node is the SDO-server.

Prototype:

```
short COP_CreateSDO( COP_t_HANDLE boardhdl,
                     BYTE         node_no,
                     BYTE         sdo_no,
                     WORD         clientCANid,
                     WORD         serverCANid );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
node_no	(in)	Node-ID of the node allocated to the SDO (between 1 and 127).
sdo_no	(in)	Number of the SDO This value must always be COP_k_USERDEFINED_SDO, because a maximum of 2 SDOs per node can be managed and the firmware for each registered node sets up the default-SDO COP_k_DEFAULT_SDO automatically.
clientCANid	(in)	Identifier of the CAN-object that the SDO-Client uses for the request to the Server
serverCANid	(in)	Identifier of the CAN-object that the SDO-Server uses for the response

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Operation could not be entered in the transmit-command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Success
COP_k_NOT_FOUND	No node registered with the stated node-ID
COP_k_CAL_ERR	Server CAN-ID not available
COP_k_IV	Invalid parameter value
COP_k_SDO_RUNNING	SDO transfer currently running, thus the user-defined SDO may not be altered at the moment

5.3.5 COP_GetSDOInfo

Description: COP_GetSDOInfo delivers the properties of a client service data object for SDO-communication with a registered node. In order to change these properties call function COP_CreateSDO().

Prototype:

```
short COP_GetSDOInfo( COP_t_HANDLE boardhdl,
                      BYTE         node_no,
                      BYTE         sdo_no,
                      WORD*        clientCANid,
                      WORD*        serverCANid );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
node_no	(in)	Node-ID of the node allocated to the SDO (between 1 and 127).
sdo_no	(in)	Number of the SDO The following values are permitted: COP_k_DEFAULT_SDO for the automatically established SDO COP_k_USERDEFINED_SDO for the additional user defined SDO.
clientCANid	(out)	Set identifier of the CAN-object that the SDO-Client uses for the request to the Server
serverCANid	(out)	Set identifier of the CAN-object that the SDO-Server uses for the response

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Operation could not be entered in the transmit-command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Success
COP_k_NOT_FOUND	No node registered with the stated node-ID
COP_k_IV	Invalid parameter value

5.3.6 COP_SetSDOTimeOut

Description: COP_SetSDOTimeOut defines the delay time that determines how long the CANopen Master Firmware is to wait for the individual response of the SDO-server with a running SDO-transfer.
The preset value is 200 ms.
Information on the value range and the resolution of the w_timeout is given in Appendix H - Timer resolutions and value ranges.

Prototype:

```
short COP_SetSDOTimeOut( COP_t_HANDLE boardhdl,
                         WORD         w_timeout );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
w_timeout	(in)	New value for the delay time in milliseconds

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Operation could not be entered in the transmit-command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Success
COP_k_CAL_ERR	General error of the Master Firmware

5.3.7 COP_DefSyncObj

Description: COP_DefSyncObj defines the cycle time for the system service of the synchronization object.
The preset value for the cycle time is 1000 ms.
When this function is called, the transmit mode of the synchronization object is ended. Therefore COP_EnableSync() must always follow.
The value set here first applies to all CAN lines of a board, as the firmware only knows one time base. In order to still be able to configure different cycle times for the different CAN lines of a board, use the function COP_SetSyncDivisor().
Information on the value range and the resolution of the sync_period is given in Appendix H - Timer resolutions and value ranges.

Prototype:

```
short COP_DefSyncObj(
                    COP_t_HANDLE boardhdl,
                    WORD         sync_period,
                    WORD         sync_window,
                    BYTE         CounterOverflow );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
sync_period	(in)	New value for the cycle time in milliseconds. The cycle time defines the time-lag between successive synchronization objects. Value range is 2 <= sync_period <= 65280.
sync_window	(in)	Reserved It is recommended to use the same value as sync_period, since 0 is invalid.
CounterOverflow	(in)	Maximum value of the CANopen sync counter. Value range is 2 <= CounterOverflow <= 240 and 0 deactivating the sync counter. If the sync counter is activated, the sync message is sent with an additional data byte that is incremented for each transmission, and that is reset to 1 every time the maximum value is overstepped. With deactivated sync counter, the sync message is sent without data byte as usual.

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Operation could not be entered in the transmit-command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Success
COP_k_CAL_ERR	General error of the Master Firmware
COP_k_IV	Invalid parameter value

5.3.8 COP_SetSyncDivisor

Description: With COP_SetSyncDivisor the divisor for the cycle time for the system service of the synchronization object of the Master Firmware is set.
The firmware works with the same time basis for all CAN lines of a board. In order to still be able to implement different cycle times for the CAN lines, it is possible to define via the divisor at what internal cycle a SYNC object is actually to be transmitted. If, for example, a firmware-global Sync cycle time of 5 ms has been set with the function COP_DefSyncObj(), the SYNC object appears on all CAN lines at the same time at intervals of 5 ms (if the functionality for the corresponding CAN line has been activated with COP_EnableSync()). An intended cycle time of 10 ms is now achieved with a divisor value of 2, therefore the firmware effectively only transmits every second SYNC object.
To handle different cycle times for the CAN lines, the greatest common divisor of both cycle times must first be transmitted as the cycle time to COP_DefSyncObj(). Then the divisor of the intended cycle time with the firmware cycle time as the divisor is defined for each CAN line.

Example: CAN line 1 intended cycle time: 100ms
CAN line 2 intended cycle time: 30ms
Largest common divisor: 10 (sync_period)
Divisor for CAN line 1: 100ms / 10ms = 10 (divisor)
Divisor for CAN line 2: 30ms / 10 ms = 3 (divisor)

Prototype:

```
short COP_SetSyncDivisor( COP_t_HANDLE boardhdl,
                          BYTE         divisor );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN board/line combination
divisor	(in)	Divisor for the intended cycle time for the respective CAN line with the Master Firmware cycle time set via COP_DefSyncObj.

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Operation could not be entered in transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Success
COP_k_CAL_ERR	General error of the Master Firmware
COP_k_IV	Invalid parameter value

5.3.9 COP_GetSyncInfo

Description: COP_GetSyncInfo delivers the properties of the system service of the synchronization object.
In order to change these properties call function COP_DefSyncObj() and COP_SetSyncDivisor().

Prototype:

```
short COP_GetSyncInfo( COP_t_HANDLE boardhdl,
                       WORD*        sync_period,
                       WORD*        sync_window,
                       BYTE*        CounterOverflow,
                       BYTE*        divisor );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
sync_period	(out)	Set value for the cycle time in milliseconds. The cycle time defines the time-lag between successive synchronization objects.
sync_window	(out)	Reserved
CounterOverflow	(out)	Set maximum value of the CANopen sync counter. 0 deactivates the sync counter. If the sync counter is activated, the sync message is sent with an additional data byte that is incremented for each transmission, and that is reset to 1 every time the maximum value is overstepped. With deactivated sync counter, the sync message is sent without data byte as usual.
divisor	(out)	Set divisor for the intended cycle time for the respective CAN line with the Master Firmware cycle time set via COP_DefSyncObj.

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Operation could not be entered in the transmit-command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Success

5.3.10 COP_EnableSync

Description: COP_EnableSync starts cyclic transmission of the synchronization object by the Master Firmware.
Is this function being called prior to calling COP_DefSyncObj() the predefined cycle time of 1000ms will be applied.
With COP_CheckSync() the Client-application can query transmission of a Sync object. If a Callback is registered for the Sync-Queue, that function is automatically called as soon as the Master has transmitted a Sync-object.

Prototype:

```
short COP_EnableSync( COP_t_HANDLE boardhdl,
                      BYTE         mode );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
mode	(in)	Operating mode of the function. The following values are permitted: COP_k_SINGLE_LINE Function only aimed at the CAN line implicitly coded in boardhdl COP_k_ALL_LINES Function aimed at all CAN lines of the board coded in boardhdl

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Operation could not be entered in the transmit-command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Success
COP_k_CAL_ERR	General error of the Master Firmware
COP_k_IV	Unauthorised parameter value

5.3.11 COP_DisableSync

Description: COP_DisableSync ends cyclic transmission of the synchronization object by the Master Firmware.

Prototype:

```
short COP_DisableSync( COP_t_HANDLE boardhdl,
                       BYTE         mode );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
mode	(in)	Operating mode of the function. The following values are permitted: COP_k_SINGLE_LINE Function only aimed at the CAN line implicitly coded in boardhdl COP_k_ALL_LINES Function aimed at all CAN lines of the board coded in boardhdl

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Operation could not be entered in the transmit-command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Success
COP_k_CAL_ERR	General error of the Master Firmware
COP_k_IV	Unauthorized parameter value

5.3.12 COP_InitTimeStampObj

Description: With `COP_InitTimeStampObj` the current time for the system service of the central time information (TimeStamp Object) is transfered.
This function must be called before the time service is activated by means of `COP_StartStopTSObj()`.

Prototype:

```
short COP_InitTimeStampObj( COP_t_HANDLE boardhdl,
                            DWORD        ms,
                            WORD         days );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
ms	(in)	Time: Milliseconds after midnight
days	(in)	Date: Days since January 1st, 1984

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Operation could not be entered in the transmit-command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Success
COP_k_IV	Invalid time specification
COP_k_BSY	Timestamp queue is full

5.3.13 COP_StartStopTSObj

Description: With `COP_StartStopTSObj` cyclic transmission of the central time information (TimeStamp Object) by the Master Firmware is started or stopped.
Information on the value range and the resolution of the cycle time `cycle` is given in Appendix H - Timer resolutions and value ranges.

Prototype:

```
short COP_StartStopTSObj( COP_t_HANDLE boardhdl,
                          BYTE         startstop,
                          WORD         cycle );
```

Parameters:

Parameter	Dir.	Explanation
`boardhdl`	(in)	Handle of the CAN-board/line combination
`startstop`	(in)	Switch for the TimeStamp Object. The following values are permitted: `COP_k_TS_START` Start transmission `COP_k_TS_STOP` End transmission
`cycle`	(in)	Cycle time in milliseconds. The cycle time defines the interval of consecutive time information.

Return values:

Return value	Description
`BER_k_ERR`	Handle invalid
`BER_k_NOT_SENT`	Operation could not be entered in the transmit-command queue
`BER_k_TIMEOUT`	No response from Master Firmware
`COP_k_OK`	Success
`COP_k_CAL_ERR`	General error of the Master Firmware

5.3.14 COP_GetTimeStampObj

Description: COP_GetTimeStampObj delivers the properties and the current value of the system service of the central time information (TimeStamp Object).
In order to change these properties call function COP_InitTimeStampObj().

Prototype:

```
short COP_GetTimeStampObj( COP_t_HANDLE boardhdl,
                           BYTE*        startstop,
                           WORD*        cycle,
                           DWORD*       ms,
                           WORD*        days );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
startstop	(out)	Set state of the TimeStamp service. The following values are possible: COP_k_TS_START Transmission active COP_k_TS_STOP Transmission disabled
cycle	(out)	Set cycle time in milliseconds. The cycle time defines the interval of consecutive time information.
ms	(out)	Current time: Milliseconds after midnight
days	(out)	Current date: Days since January 1st, 1984

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Operation could not be entered in the transmit-command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Success

5.4 CANopen communication

The functions for the CANopen communication are used for the direct information exchange with the individual CANopen devices.

5.4.1 COP_ReadPDO

Description: COP_ReadPDO reads the data of a process data object (PDO) received by the Master Firmware from the RPDO-queue.
With the exception of the four **Predefined Connection Set** RPDOs a PDO can only be received if it had been set up beforehand with function COP_CreatePDO(). The combination of node-ID and pdo# (node_no / pdo_no) serves as its unique identification. The actual properties of a set up PDO might be queried any time by calling COP_GetPDOInfo().

Prototype:

```
short COP_ReadPDO( COP_t_HANDLE boardhdl,
                   BYTE*        node_no,
                   BYTE*        pdo_no,
                   BYTE*        rxlen,
                   BYTE*        rxdata,
                   BYTE*        SyncCounter );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
node_no	(out)	Number of the node that has transmitted the PDO (between 1 and 127)
pdo_no	(out)	Number of the PDO, beginning with 1
rxlen	(out)	Number of valid bytes of rxdata
rxdata	(out)	Address of an 8 byte buffer for the received PDO-data.
SyncCounter	(out)	Value of the sync object's sync counter on receipt of the PDO. For an explanation of the sync counter see COP_DefSyncObj()

Return values:

Return value	Description
BER_k_ERR	Handle invalid
COP_k_OK	Success
COP_k_QUEUE_EMPTY	No new PDOs available
COP_k_IV	NULL pointer as parameter

5.4.2 COP_ReadPDO_S

Description: COP_ReadPDO_S reads out the data of a process data object (PDO) received by the Master Firmware from the RPDO-Queue. COP_ReadPDO_S works in the same way as COP_ReadPDO(). In contrast to COP_ReadPDO(), however, the function returns the PDO-data as a structure.

Prototype:

```
short COP_ReadPDO_S( COP_t_HANDLE  boardhdl,
                     COP_t_RX_PDO* sp_pdo );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
sp_pdo	(out)	Pointer to a buffer provided by the Client-application of data type COP_t_RX_PDO, which accepts the PDO data

COP_t_RX_PDO Alignment: 1 byte

Field	Type	Meaning
node_no	BYTE	Number of the node that the PDO has transmitted (between 1 and 127)
pdo_no	BYTE	Number of the PDO, beginning with 1
length	BYTE	Number of valid bytes of a_data
SyncCounter	BYTE	Value of the sync object's sync counter on receipt of the PDO. For an explanation of the sync counter see COP_DefSyncObj()
a_data[8]	BYTE[]	Received PDO-data

Return values:

Return value	Description
BER_k_ERR	Handle invalid
COP_k_OK	Success
COP_k_QUEUE_EMPTY	No new PDOs available
COP_k_IV	NULL pointer as parameter

5.4.3 COP_RequestPDO

Description: COP_RequestPDO initiates a request for a process data object (PDO).
The data of the requested PDO are then received via the RPDO-Queue (read-out as usual with COP_ReadPDO() or COP_ReadPDO_S()).

Prototype:

```
short COP_RequestPDO( COP_t_HANDLE boardhdl,
                      BYTE         node_no,
                      BYTE         pdo_no );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
node_no	(in)	Number of the registered node that is to transmit the PDO (between 1 and 127)
pdo_no	(in)	Number of the PDO, beginning with 1

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Operation could not be entered in the transmit-command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Success
COP_k_NOT_FOUND	No node registered with the stated node-ID
COP_k_BSY	CAN transmit-queue full
COP_k_CAL_ERR	General error of the Master Firmware
COP_k_IV	Invalid parameter value

5.4.4 COP_WritePDO

Description: COP_WritePDO writes the data of a process data object to be transmitted by the Master Firmware into the TPDO-Queue. With the exception of the four **Predefined Connection Set** TPDOs a PDO can only be transmitted if it had been set up beforehand with function COP_CreatePDO(). The combination of node-ID and pdo# (node_no / pdo_no) serves as its unique identification. The actual properties of a set up PDO might be queried any time by calling COP_GetPDOInfo().

PDOs with incorrect parameters (node_no, pdo_no) are rejected by the firmware. Via the Event-Queue a corresponding message of type COP_k_WPDO_EVT **is then returned (read-out of the Event-Queue with COP_GetEvent()). Since the corresponding parameter check is performed in the firmware, and since the API-function COP_WritePDO() is working unconfirmedly for the purpose of performance gain, the function call returns COP_k_OK rather than COP_k_IV in this case.**

Prototype:

```
short COP_WritePDO( COP_t_HANDLE boardhdl,
                    BYTE         node_no,
                    BYTE         pdo_no,
                    BYTE*        txdata );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
node_no	(in)	Number of the node to which the PDO is to be transmitted (between 0 and 127)
pdo_no	(in)	Number of the TPDO (beginning with 1)
txdata	(in)	Address of an 8 byte buffer for the PDO-data to be transmitted.

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Operation could not be entered in transmit-PDO-queue
COP_k_OK	Success
COP_k_IV	Invalid parameter value
COP_k_BSY	Transmit-PDO-queue of the firmware is full

5.4.5 COP_WritePDO_S

Description: COP_WritePDO_S writes the data of a process data object to be transmitted by the Master Firmware into the TPDO-Queue. COP_WritePDO_S works in the same way as COP_WritePDO(). In contrast to COP_WritePDO(), however, the function accepts the PDO-data as a structure.

PDOs with incorrect parameters (node_no, pdo_no) are rejected by the firmware. Via the Event-Queue a corresponding message of type COP_k_WPDO_EVT **is then returned (read-out of the Event-Queue with COP_GetEvent()). Since the corresponding parameter check is performed in the firmware, and since the API-function COP_WritePDO() is working unconfirmedly for the purpose of performance gain, the function call returns COP_k_OK rather than COP_k_IV in this case.**

Prototype:

```
short COP_WritePDO_S( COP_t_HANDLE  boardhdl,
                      COP_t_TX_PDO* sp_pdo );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
sp_pdo	(in)	Pointer to a buffer provided by the Client-application of data type COP_t_TX_PDO, which contains the PDO data

COP_t_TX_PDO Alignment: 1 byte

Field	Type	Meaning
node_no	BYTE	Number of the node to which the PDO is to be transmitted (between 0 and 127)
pdo_no	BYTE	Number of the TxPDO (beginning with 1)
a_data[8]	BYTE[]	PDO-data to be transmitted

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Operation could not be entered in transmit-PDO-Queue
COP_k_OK	Success
COP_k_IV	Invalid parameter value or NULL pointer
COP_k_BSY	Transmit-PDO-queue of the firmware is full

5.4.6 COP_ReadSDO

Description: COP_ReadSDO reads out the contents of an Object Dictionary entry from a node.
The OD-entry is addressed via idx and subidx. The service data object to be used is to be specified in sdo_no.
The function works synchronously, i.e. the call only returns to the Client-application when the full (segmented where required) SDO-Transfer is finished.

Prototype:

```
short COP_ReadSDO( COP_t_HANDLE boardhdl,
                   BYTE         node_no,
                   BYTE         sdo_no,
                   BYTE         mode,
                   WORD         idx,
                   BYTE         subidx,
                   DWORD*       rxlen,
                   BYTE*        rxdata,
                   DWORD*       abortcode );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
node_no	(in)	Number of the node, from whose object dictionary an entry is to be read (between 1 and 127)
sdo_no	(in)	Number of the service data object to be used. The following values are permitted: COP_k_DEFAULT_SDO means that the default-SDO for the node, which the firmware automatically creates, is to be used. COP_k_USERDEFINED_SDO means that the additional SDO, which must have been created previously by COP_CreateSDO(), is to be used.
mode	(in)	Definition of the SDO-transmission protocol. With more than 4 bytes of data to be transmitted, the reference data are generally divided into 7-byte segments. The following values are permitted: COP_k_NO_BLOCKTRANSFER Use of the (conventional) Domain-protocol, with which the receipt of each segment is confirmed. COP_k_BLOCKTRANSFER Use of the block transfer protocol, with which confirmation is given only after max. 127 segments. The implementation of the block transfer is optional and is not supported by every node.
idx	(in)	MainIndex of the object dictionary entry to be read

subidx	(in)	SubIndex of the object dictionary entry to be read
rxlen	(in/out)	Size of the receive buffer rxdata. If the receive buffer is not sufficient, no separate error code is returned (in the case of success, for example, the return value is COP_k_OK), and the SDO-Transfer is not aborted but the number of required bytes is returned to the Client-application in this parameter. An internal buffer overrun is prevented by the excess bytes received being rejected when the buffer capacity is exhausted.
rxdata	(out)	Address of a suffiently large buffer for the object dictionary data received.
abortcode	(out)	Possible Abort-Code of the SDO-Transfer (optional parameter) In case of an abort, COP_k_ABORT is returned as return value.

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_MEM_ALLOC_ERR	Internal data structures or operating system-objects could not be created
BER_k_SDO_THREAD_ERR	Error with control of the SDO-Thread
BER_k_PC_MC_COMM_ERR	Communication link with the CAN board is disturbed
COP_k_OK	Success
COP_k_NOT_FOUND	No node registered with the stated node-ID
COP_k_IV	Invalid parameter value
COP_k_ABORT	SDO-Transfer aborted by one of the two partners. Error-code included in abortcode
COP_k_QUEUE_EMPTY	No SDO-response of the Master Firmware
COP_k_TIMEOUT	No response from the device, so SDO-Transfer aborted by the Master Firmware
COP_k_SDO_RUNNING	Previous SDO transfer is not finished yet

5.4.7 COP_WriteSDO

Description: COP_WriteSDO writes data into an object dictionary entry of a node.
The OD-entry is addressed via `idx` and `subidx`. The service data object to be used is to be specified in `sdo_no`.
The function works synchronously, i.e. the call returns to the Client-application only when the full (segmented where required) SDO-Transfer is finished.

Prototype:

```
short COP_WriteSDO( COP_t_BOARD boardhdl,
                    BYTE        node_no,
                    BYTE        sdo_no,
                    BYTE        mode,
                    WORD        idx,
                    BYTE        subidx,
                    DWORD       txlen,
                    BYTE*       txdata,
                    DWORD*      abortcode );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
node_no	(in)	Number of the node whose object dictionary entry is to be written to (between 1 and 127)
sdo_no	(in)	Number of the service data object to be used. The following values are permitted: COP_k_DEFAULT_SDO means that the default-SDO for the node that the firmware automatically creates is to be used. COP_k_USERDEFINED_SDO means that the additional SDO, which must have been created previously by means of COP_CreateSDO(), is to be used.
mode	(in)	Definition of the SDO-transmission protocol. With more than 4 bytes of data to be transmitted, the reference data are generally divided into 7-byte segments. The following values are permitted: COP_k_NO_BLOCKTRANSFER Use of the (conventional) Domain-protocol, with which the receipt of each segment is confirmed. COP_k_BLOCKTRANSFER Use of the block transfer protocol, with which confirmation is given only after max. 127 segments. The implementation of the block transfer is optional and is not supported by every node.
idx	(in)	MainIndex of the object dictionary entry to be written
subidx	(in)	SubIndex of the object dictionary entry to be written

txlen	(in)	Number of bytes to be transmitted, i.e. size of the transmit buffer txdata
txdata	(in)	Address of the buffer containing the object dictionary data.
abortcode	(out)	Possible Abort-Code of the SDO-Transfer (optional parameter) In case of an abort, COP_k_ABORT is returned as return value.

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_MEM_ALLOC_ERR	Internal data structures or operating system-objects could not be created
BER_k_SDO_THREAD_ERR	Error with control of the SDO-Thread
BER_k_PC_MC_COMM_ERR	Communication link with the CAN board is disturbed
COP_k_OK	Success
COP_k_NOT_FOUND	No node registered with the stated node-ID
COP_k_IV	Invalid parameter value
COP_k_ABORT	SDO-Transfer aborted by one of the two partners. Error-code included in abortcode
COP_k_QUEUE_EMPTY	No SDO-response of the Master Firmware
COP_k_TIMEOUT	No response from the device, so SDO-Transfer aborted by the Master Firmware
COP_k_SDO_RUNNING	Previous SDO transfer is not finished yet

5.4.8 COP_PutSDO

Description: Initiates reading or writing of a service data object by placing an SDO-operation in the transmit-SDO-Queue.
The function is not blocking (asynchronous) and therefore returns to the Client-application immediately. After termination of the SDO-Transfer the result must be read with `COP_GetSDO()`. The optional Event Handle `h_Event` can be used to wait synchronously for the end of the SDO-Transfer by means of `WaitForSingleObject()`.

Prototype:

```
short COP_PutSDO( COP_t_HANDLE boardhdl,
                  BYTE         node_no,
                  BYTE         sdo_no,
                  BYTE         mode,
                  BYTE         rwAccess,
                  WORD         idx,
                  BYTE         subidx,
                  DWORD        length,
                  BYTE*        data,
                  HANDLE       h_Event );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
node_no	(in)	Number of the node whose object dictionary is to be accessed (between 1 and 127)
sdo_no	(in)	Number of the service data object to be used. The following values are permitted: COP_k_DEFAULT_SDO means that the default-SDO for the node, which the firmware automatically creates, is to be used. COP_k_USERDEFINED_SDO means that the additional SDO, which must have been created previously by means of `COP_CreateSDO()`, is to be used.
mode	(in)	Definition of the SDO-transmission protocol. With more than 4 bytes of data to be transmitted, the reference data are generally divided into 7-byte segments. The following values are permitted: COP_k_NO_BLOCKTRANSFER Use of the (conventional) Domain-protocol, with which the receipt of each segment is confirmed. COP_k_BLOCKTRANSFER Use of the block transfer protocol, with which confirmation is given only after max. 127 segments. The implementation of the block transfer is optional and is not supported by every node.

rwAccess	(in)	Transmission direction (read/write). The following values are permitted: COP_k_SDO_DOWNLOAD Writing object dictionary access (the data are transmitted to the node by the Master Firmware). COP_k_SDO_UPLOAD Reading object dictionary access (the data are transmitted from the node to the Master Firmware)
idx	(in)	MainIndex of the object dictionary entry
subidx	(in)	SubIndex of the object dictionary entry
length	(in)	For COP_k_SDO_DOWNLOAD: size of the buffer data.
data	(in)	For COP_k_SDO_DOWNLOAD: Address of the buffer containing the data to be transmitted.
h_Event	(in)	Optional Handle of an operating system-Event. The Windows-Event must be created by the Client-application by means of CreateEvent() in the initial state non-signaled.

Return values:

Return value	Description
BER_k_ERR	Handle invalid
COP_k_OK	Success
COP_k_IV	Invalid parameter value
COP_k_SDO_RUNNING	SDO-operation already in progress but not yet ended or result not yet obtained by means of COP_GetSDO().

5.4.9 COP_GetSDO

Description: Reads the result of an SDO-transfer previously initialized with `COP_PutSDO()` from the receive-SDO-Queue.
The function is not blocking (asynchronous) and therefore returns to the Client-application immediately. The Event Handle transferred with `COP_PutSDO()` can be used to wait expressly for the end of the SDO-Transfer without having to poll with `COP_GetSDO`.

Prototype:

```
short COP_GetSDO( COP_t_HANDLE boardhdl,
                  DWORD*       length,
                  BYTE*        data,
                  DWORD*       abortcode );
```

Parameters:

Parameter	Dir.	Explanation
`boardhdl`	(in)	Handle of the CAN-board/line combination
`length`	(in/out)	Size of the receive buffer `data` or number of bytes transmitted
`data`	(out)	Address of a buffer for the object dictionary data received
`abortcode`	(out)	Possible Abort-Code of the SDO-Transfer (optional parameter) In case of an abort, `COP_k_ABORT` is returned as return value.

Return values:

Return value	Description
`BER_k_ERR`	Handle invalid
`BER_k_SDO_THREAD_ERR`	SDO transfer cancelled using `COP_CancelSDO()`
`BER_k_TIMEOUT`	No response from Master Firmware
`BER_k_PC_MC_COMM_ERR`	Communication link with the CAN board is disturbed
`BER_k_DATA_CORRUPT`	Corrupt data received, Communication path (USB, Ethernet) disturbed, automatic retry
`COP_k_OK`	Success
`COP_k_IV`	Invalid parameter value
`COP_k_NOT_FOUND`	No node registered with the stated node-ID
`COP_k_SDO_RUNNING`	SDO transfer currently running, the approximate progress in ‰ is included in `length`
`COP_k_ABORT`	SDO transfer aborted by one of the two partners. Error-code included in `abortcode`
`COP_k_QUEUE_EMPTY`	No SDO-response of the Master Firmware
`COP_k_TIMEOUT`	No response from the device, so SDO-Transfer aborted by the Master Firmware

5.4.10 COP_CancelSDO

Description: With COP_CancelSDO a running SDO-operation, which was previously initiated with COP_PutSDO(), is aborted.

Prototype:

```
short COP_CancelSDO( COP_t_HANDLE boardhdl,
                     BYTE         node_no,
                     BYTE         sdo_no,
                     WORD         idx,
                     BYTE         subidx );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
node_no	(in)	Number of the node whose object dictionary is being accessed (between 1 and 127)
sdo_no	(in)	Number of the service data object used for the current transfer. The following values are permitted: COP_k_DEFAULT_SDO COP_k_USERDEFINED_SDO
idx	(in)	MainIndex of the current SDO-Transfer
subidx	(in)	SubIndex of the current SDO-Transfer

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Operation could not be entered in the transmit-command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Success
COP_k_IV	Invalid parameter value
COP_k_NOT_FOUND	No node registered with the stated node-ID

5.4.11 COP_GetEmergencyObj

Description: COP_GetEmergencyObj reads an emergency object from the EMCY-Queue and returns it subdivided into Errorvalue, Errorregister and Errordata.

Prototype:

```
short COP_GetEmergencyObj(
                        COP_t_HANDLE boardhdl,
                        BYTE*        node_no,
                        WORD*        err_value,
                        BYTE*        err_register,
                        BYTE*        err_data );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
node_no	(out)	Number of the node that has issued the error message (between 1 and 127)
err_value	(out)	Error code of the error message. The error codes are standardized according to CiA-301.
err_register	(out)	Contents of the device error register
err_data	(out)	Address of a 5-byte buffer for the manufacturer-specific error field

Return values:

Return value	Description
BER_k_ERR	Handle invalid
COP_k_OK	Success
COP_k_QUEUE_EMPTY	No new emergency objects available
COP_k_IV	NULL pointer as parameter

5.4.12 COP_GetEmergencyObj_S

Description: COP_GetEmergencyObj_S reads an emergency object from the EMCY-Queue.
COP_GetEmergencyObj_S works in the same way as COP_GetEmergencyObj(). In contrast to COP_GetEmergencyObj(), however, the function returns the alarm message as a structure.

Prototype:

```
short COP_GetEmergencyObj_S(
                COP_t_HANDLE          boardhdl,
                COP_t_EMERGENCY_OBJ*  sp_emergency );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
sp_emergency	(out)	Pointer to a buffer provided by the Client-application of data type COP_t_EMERGENCY_OBJ, which receives thev alarm message

COP_t_EMERGENCY_OBJ Alignment: 1 byte

Field	Type	Meaning
err_value	WORD	Error code of the error message. The error codes are standardized according to DS-301 (chapter 7.2.7).
err_reg	BYTE	Contents of the device error register
err_data[5]	BYTE[]	Manufacturer-specific error field
node_no	BYTE	Number of the node that has issued the error message (between 1 and 127)

Return values:

Return value	Description
BER_k_ERR	Handle invalid
COP_k_OK	Success
COP_k_QUEUE_EMPTY	No new emergency objects available
COP_k_IV	NULL pointer as parameter

5.4.13 COP_CheckSync

Description: COP_CheckSync checks whether a Sync-object has been signaled by the CANopen-Master Firmware.
Every time the Master Firmware has transmitted the Sync-object, an entry is made in the Sync-Queue. The Client application is thus able to react to a Sync-event, for example in order to read out cylic synchronous PDOs.

Prototype:

```
short COP_CheckSync( COP_t_HANDLE boardhdl,
                     BYTE*        SyncCounter );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
SyncCounter	(out)	Value of the sync counter on sending of the sync object. For an explanation of the sync counter see COP_DefSyncObj()

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_PC_MC_COMM_ERR	Communication link with the CAN board is disturbed
COP_k_OK	Sync carried out
COP_k_QUEUE_EMPTY	No Sync-event available

5.4.14 COP_GetEvent

Description: COP_GetEvent reads a network- or Master Firmware-event from the Event-Queue.
The event can have various meanings: network-Event (e.g. node failed), local Status-Event (e.g. CAN-error, overruns), Queue-overrun; WritePDO-Event (e.g. incorrect parameter - PDO rejected) or Flying Master-Event (e.g. network mastership lost).

Prototype:

```
short COP_GetEvent( COP_t_HANDLE boardhdl,
                    BYTE*        evt_type,
                    BYTE*        evt_data1,
                    BYTE*        evt_data2,
                    BYTE*        evt_data3 );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
evt_type	(out)	Type of event. The following values are possible: COP_k_NMT_EVT Network event COP_k_DLL_EVT Local event of the Data Link Layer COP_k_WPDO_EVT Event triggered by a transmit-PDO operation COP_k_RPDO_EVT Receive-PDO event COP_k_QUEUE_OVRUN_EVT Overrun of one of the three receive queues (EMCY, RPDO, Event) COP_k_FLY_EVT Event connected with Flying Master additional functionality
evt_data1	(out)	Additional information on the event
evt_data2	(out)	Additional information on the event
evt_data3	(out)	Additional information on the event

Depending on the contents of the parameter evt_type additional information on the event is given in the other three parameters evt_dataX. These are listed in the following tables.

*evt_type == COP_k_NMT_EVT

evt_data1	evt_data2	evt_data3
Cause of event: COP_k_NMT_GUARDERR Guard error. Device has not responded or the signaled node-state is unexpected COP_k_NMT_BOOTIND Device has sent Bootup message COP_k_NMT_HEARTBEATERR Heartbeat error. Device has transmitted nothing or signaled node-state is unexpected	Node-ID of the device involved	Signaled node state: COP_k_NS_STOPPED, COP_k_NS_OPERATIONAL, COP_k_NS_PREOPERATIONAL and COP_k_NS_DISCONNECTED (see also COP_GetNodeState)

*evt_type == COP_k_DLL_EVT

evt_data1	evt_data2	evt_data3
Statusflags of the Data Link Layer, bit coded: COP_k_DLL_COVR Overrun of the receive queue of the CAN-Controller COP_k_DLL_BOFF CAN-Controller in the BusOff state COP_k_DLL_ESET Error Status Bit of the CAN-Controller set COP_k_DLL_ERESET Error Status Bit of the CAN-Controller reset COP_k_DLL_RXOVR Overrun of the firmware-internal receive queue COP_k_DLL_TXOVR Overrun of the firmware-internal transmit queue	unused	unused

*evt_type == COP_k_WPDO_EVT
*evt_type == COP_k_RPDO_EVT

evt_data1	evt_data2	evt_data3
Event in connection with the PDO-Queues: COP_k_ERR_PDO_IV Invalid parameter in the transmit-PDO-operation (triggered by COP_WritePDO) COP_k_ERR_PDO_OVR Overrun of the firmware-internal transmit- or receive queue. This means that the corresponding PDO was lost. In addition to this event, a COP_k_DLL_EVT with *evt_data1 = COP_k_DLL_RXOVR or = COP_k_DLL_TXOVR is also generated.	Node-ID of the device involved (node_no)	Number of the PDO involved (pdo_no)

*evt_type == COP_k_QUEUE_OVRUN_EVT

evt_data1	evt_data2	evt_data3
Overrun counter of the EMCY-Queue, i.e. number of lost emergency messages	Overrun counter of the RPDO-Queue, i.e. number of lost receive-PDOs	Overrun counter of the Event-Queue, i.e. number of lost events

*evt_type == COP_k_FLY_EVT

evt_data1	evt_data2	evt_data3
Event connected with the Flying Master additional functionality COP_k_FLY_MASTER Firmware has gained network mastership and is the active NMT Master COP_k_FLY_NOT_MASTER Firmware lost the negotiation and is not the active NMT Master COP_k_FLY_LOST_MASTERSHIP Firmware has lost network mastership, as it was replaced by a higher priority NMT Master COP_k_FLY_LOST_ACTIVE_MASTER Active NMT Master failed. Now negotiation of the new NMT Master begins with all other potential masters in the network. COP_k_FLY_WAIT_BUSCONNECTION CANopen Master Firmware is not on CAN COP_k_FLY_NEGOTIATION_RUNNING Negotiation of network mastership with other potential masters is running	not used	not used

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_PC_MC_COMM_ERR	Communication link with the CAN board is disturbed
COP_k_OK	Success
COP_k_QUEUE_EMPTY	No new entries available
COP_k_IV	NULL pointer as parameter

5.5 LMT services

This so-called LMT service is required to set the parameters of the CANopen network for devices without direct user interface (such as DIP-switch). Not all devices support the LMT service – not least because it has been superseded by LSS.

5.5.1 COP_LMT_ConfigNode

Description: With `COP_LMT_ConfigNode` the node-ID and baudrate of a CANopen device are set using the LMT service.
The target device is unmistakeably identified worldwide via its LMT-address, which is made up of manufacturer's name, product code and serial number.
In this function a whole set of LMT-commands is issued in the following sequence:

(1) SwitchModeSelective using the parameters `sz_mname`, `sz_pname` and `sz_sno`.
(2) ConfigureModuleID using the parameter `new_node_no`.
(3) ConfigureBitTimingParameters using the parameter `new_baudrate`.
(4) StoreConfiguration.

When this COP_LMT_ConfigNode() function is called, the firmware is reset internally. Then a complete reinitialization of the CANopen Master Firmware is therefore required in accordance with Fig. 2-1, beginning with COP_InitInterface().

Prototype:

```
short COP_LMT_ConfigNode(
                        COP_t_HANDLE boardhdl,
                        char*        sz_mname,
                        char*        sz_pname,
                        char*        sz_sno,
                        BYTE         new_node_no,
                        WORD         access_baudrate,
                        WORD         new_baudrate );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
sz_mname	(in)	Manufacturer's name of the device to be configured (LMT-Address.manufacturer-name) 7 characters
sz_pname	(in)	Product code of the device to be configured (LMT-Address.product-name) 7 characters
sz_sno	(in)	Serial number of the device to be configured (LMT-Address.serial-number) 14 characters
new_node_no	(in)	New node-ID (to be set) between 1 and 127
access_baudrate	(in)	Baudrate for the CAN-communication during the LMT-configuration process. The following values are permitted: COP_k_10_KB COP_k_20_KB COP_k_50_KB COP_k_100_KB COP_k_125_KB COP_k_250_KB COP_k_500_KB COP_k_800_KB COP_k_1000_KB
new_baudrate	(in)	New baudrate (to be set)

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Operation could not be entered in the transmit-command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Success
COP_k_NO	General error of the Master Firmware
COP_k_TIMEOUT	No response from the device
COP_k_IV	Invalid parameter value or NULL pointer

5.5.2 COP_LMT_GetAddress

Description: With `COP_LMT_GetAddress` the LMT address of a CANopen device is read out.
A CANopen device that supports the LMT-service can be unmistakeably identified worldwide via its LMT-address, which is made up of manufacturer's name, product code and serial number.
In this function a whole set of LMT-commands is issued in the following sequence:

(1) SwitchModeGlobal for activating the LMT service.
(2) InquireManufacturerName to inquire the manu facturer's name `sz_mname`.
(3) InquireProductName to inquire the product code `sz_pname`
(4) InquireSerialNumber to inquire the serial number `sz_sno`.
(5) SwitchModeGlobal to deactivate the LMT service.

As SwitchModeGlobal is used within this CANopen API function, it may only be used in such cases where only *one* LMT-compatible device exists in the network, as otherwise the device responses overlap and destroy each other.

Prototype:

```
short COP_LMT_GetAddress( COP_t_HANDLE boardhdl,
                          BYTE         baudrate,
                          char*        sz_mname,
                          char*        sz_pname,
                          char*        sz_sno );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
baudrate	(in)	Baudrate for the CAN-communication during the LMT-configuration process. The following values are permitted: COP_k_10_KB COP_k_20_KB COP_k_50_KB COP_k_100_KB COP_k_125_KB COP_k_250_KB COP_k_500_KB COP_k_800_KB COP_k_1000_KB
sz_mname	(out)	Pointer to an 8-byte buffer provided by the Client-application, which receives the read-out manufacturer's name of the device (LMT-Address.manufacturer-name) 7 characters + terminating \0

sz_pname	(out)	Pointer to an 8-byte buffer provided by the Client-application, which receives the read-out product code of the device (LMT-Address.product-name) 7 characters + terminating \0
sz_sno	(out	Pointer to a 15-byte buffer provided by the Client-application, which receives the read-out serial number of the device (LMT-Address.serial-number) 14 characters + terminating \0

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Operation could not be entered in the transmit-command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Success
COP_k_NO	General error of the Master Firmware
COP_k_TIMEOUT	No response from the device

5.5.3 COP_LMT_ConfigModuleID

Description: With COP_LMT_ConfigModuleID, the node-ID of a CANopen device is configured using the LMT service.
A CANopen device that supports the LMT service can be clearly identified and addressed worldwide via its LMT address, which consists of the manufacturer name, product code and serial number.
The scope of functions is part of the similar API function COP_ConfigNode and transmits the following LMT commands:

(1) SwitchModeSelective using the parameters sz_mname, sz_pname and sz_sno.
(2) ConfigureModuleID using the parameter new_node_no.
(3) StoreConfiguration.

Prototype:

```
short COP_LMT_ConfigModuleID(
                        COP_t_HANDLE boardhdl,
                        BYTE         baudrate,
                        char*        sz_mname,
                        char*        sz_pname,
                        char*        sz_sno,
                        BYTE         new_node_no );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
baudrate	(in)	Baudrate for the CAN communication during the LMT configuration process. The following values are permitted: COP_k_10_KB COP_k_20_KB COP_k_50_KB COP_k_100_KB COP_k_125_KB COP_k_250_KB COP_k_500_KB COP_k_800_KB COP_k_1000_KB
sz_mname	(in)	Manufacturer name of the device to be configured (LMT address.manufacturer-name) 7 characters
sz_pname	(in)	Product code of the device to be configured (LMT address.product-name) 7 characters
sz_sno	(in)	Serial number of the device to be configured (LMT-Address.serial-number) 14 characters
new_node_no	(in)	New node-ID (to be configured) between 1 and 127

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Operation could not be entered in transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Success
COP_k_NO	General error of the Master Firmware
COP_k_TIMEOUT	No response from device

5.5.4 COP_LMT_IdentifyRemoteSlaves

Description: With COP_LMT_IdentifyRemoteSlaves, devices of which the manufacturer name and product code are known can be identified in the network.
If more than one device of the same type exists in the network, or if the serial numbers are not known, an individual CANopen device that supports this LMT service can be clearly identified with this function through interactive narrowing of the serial number path.
This function does not require isolated CAN communication and only transmits one LMT command:
(1) LMTIdentifyRemoteSlaves using the parameters sz_mname, sz_pname, sz_snolow and sz_snohigh.
If at least one node responds from the specified serial number path, the function returns COP_k_OK, otherwise COP_k_TIMEOUT. The delay time until the device response can be configured with the function COP_SetLSSTimeOut().

Prototype:

```
short COP_LMT_IdentifyRemoteSlaves(
                        COP_t_HANDLE boardhdl,
                        BYTE         baudrate,
                        char*        sz_mname,
                        char*        sz_pname,
                        char*        sz_snolow,
                        char*        sz_snohigh );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
baudrate	(in)	Baudrate for the CAN communication during the LMT search process. The following values are permitted: COP_k_10_KB COP_k_20_KB COP_k_50_KB COP_k_100_KB COP_k_125_KB COP_k_250_KB COP_k_500_KB COP_k_800_KB COP_k_1000_KB
sz_mname	(in)	Manufacturer name of the devices to be identified (LMT-Address.manufacturer-name) 7 characters
sz_pname	(in)	Product code of the device to be identified (LMT-Address.product-name) 7 characters
sz_snolow	(in)	Lower serial number limit of the devices to be identified (LMT-Address.serial-number) 14 characters

sz_snohigh	(in)	Upper serial number of the devices to be identified (LMT-Address.serial-number) 14 characters

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Order could not be entered in transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Success, at least one device was found
COP_k_NO	General error of the Master Firmware
COP_k_TIMEOUT	No response from device

5.6 LSS services

The so-called LSS services in accordance with CiA-305 Layer Setting Services and Protocol (LSS) are used to configure the parameters of the CANopen network for devices without a direct user interface (such as DIP-switch). Not all devices support all LSS services.

5.6.1 COP_SetLSSTimeOut

Description: `COP_SetLSSTimeOut` defines the delay time which determines how long a device response is awaited after transmitting an LSS or LMT command.
The default value for the delay time is 100 milliseconds.

Prototype:

```
short COP_SetLSSTimeOut( COP_t_HANDLE boardhdl,
                         WORD         w_timeout );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
w_timeout	(in)	New value for the delay time in milliseconds. The value range is 5 <= w_timeout <= 32767. Smaller values are rounded up internally.

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Order could not be entered in transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Success
COP_k_IV	Unauthorized parameter value

5.6.2 COP_LSS_InquireAddress

Description: `COP_LSS_InquireAddress` reads out the LSS address of a CANopen device.
A CANopen device that supports the LSS service can be clearly identified worldwide via its LSS address that corresponds with the Identity Object [1018].
In this function a complete block of LSS commands is transmitted in the following order:

(1) SwitchModeGlobal to activate the LSS service.
(2) InquireIdentityVendorID to inquire the vendor identity `VendorId`.
(3) InquireIdentityProductCode to inquire the product code `ProductCode`.
(4) InquireIdentityRevisionNumber to enquire the device revision number `RevisionNo`.
(5) InquireIdentitySerialNumber to inquire the serial number `SerialNo`.
(6) SwitchModeGlobal to deactivate the LSS service.

As SwitchModeGlobal is used in this CANopen API function, it may only be used in cases where only *one* LSS-compatible device exists in the network, as otherwise the device responses are superimposed and disturb each other.

Prototype:

```
short COP_LSS_InquireAddress(
                        COP_t_HANDLE boardhdl,
                        BYTE         baudtable,
                        BYTE         baudrate,
                        DWORD*       VendorId,
                        DWORD*       ProductCode,
                        DWORD*       RevisionNo,
                        DWORD*       SerialNo );
```

Parameters:

Parameter	Dir.	Explanation
`boardhdl`	(in)	Handle of the CAN-board/line combination
`baudtable`	(in)	Selection of the bittiming table to be used for the CAN communication during the LSS configuration process. There are two different tables:: `COP_k_BAUD_CIA` Table with the bittimings specified by CiA in CiA-301 Standard table. `COP_k_BAUD_USER` Table with userdefined bittiming settings. These must have been set before by a further API function.

baudrate	(in)	Baudrate for the CAN communication during the LSS configuration process. However, does **not** overwrite the baudrate possibly configured with COP_InitInterface() . The following values are permitted: COP_k_10_KB COP_k_20_KB COP_k_50_KB COP_k_100_KB COP_k_125_KB COP_k_250_KB COP_k_500_KB COP_k_800_KB COP_k_1000_KB
VendorId	(out)	Address of a DWORD buffer provided by the client application, which receives the vendor ID of the device that was read out
ProductCode	(out)	Address of a DWORD buffer provided by the client application, which receives the product code that was read out
RevisionNo	(out)	Address of a DWORD buffer provided by the client application, which receives the revision number that was read out
SerialNo	(out)	Address of a DWORD buffer provided by the client application, which receives the product serial number that was read out

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Order could not be entered in transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Success
COP_k_NO	General error of the Master Firmware
COP_k_TIMEOUT	No response from device
LSS_k_MEDIA_ACCESS_ERROR	CAN bus access failed
LSS_k_IV_PARAMETER	Unauthorized parameter value
LSS_k_PROTOCOL_ERR	Unauthorized device response (LSS protocol violation)
LSS_k_BSY	LSS module of the firmware already busy

5.6.3 COP_LSS_InquireNodeID

Description: `COP_LSS_InquireNodeID` inquires the node-ID of the CANopen device.
A CANopen device that supports the LSS service returns its configured node-ID, which however can also be 255. This special value indicates that no valid node-ID at all is configured and the device is in the so-called "LSS Init State".
In this function a complete block of LSS commands is transmitted in the following order:

(1a) SwitchModeGlobal to activate the LSS service, if defined.

(1b) Alternatively SwitchModeSelective to address the device using the parameters `VendorId`, `ProductCode`, `RevisionNo` and `SerialNo`.

(2) InquireNodeID to inquire the node-ID.

(3) SwitchModeGlobal to deactivate the LSS service.

If SwitchModeGlobal is used with the relevant flag, the CANopen API function can only be used in cases where only *one* LSS-compatible device exists in the network, as otherwise the device responses are superimposed and disturb each other.

Prototype:

```
short COP_LSS_InquireNodeID(
                        COP_t_HANDLE boardhdl,
                        BYTE         baudtable,
                        BYTE         baudrate,
                        BYTE         mode,
                        DWORD        VendorId,
                        DWORD        ProductCode,
                        DWORD        RevisionNo,
                        DWORD        SerialNo,
                        BYTE*        node_id );
```

Parameters:

Parameter	Dir.	Explanation
`boardhdl`	(in)	Handle of the CAN-board/line combination
`baudtable`	(in)	Selection of the bittiming table to be used for the CAN communication during the LSS configuration process. There are two different tables:: `COP_k_BAUD_CIA` Table with the bittimings specified by CiA in CiA-301 Standard table. `COP_k_BAUD_USER` Table with userdefined bittiming settings. These must have been set before by a further API function.

baudrate	(in)	Baudrate for the CAN communication during the LSS configuration process However, does **not** overwrite the baudrate possibly configured with COP_InitInterface() . The following values are permitted: COP_k_10_KB COP_k_20_KB COP_k_50_KB COP_k_100_KB COP_k_125_KB COP_k_250_KB COP_k_500_KB COP_k_800_KB COP_k_1000_KB
mode	(in)	Operating mode of the function. Possible values are: LSS_k_SET_MODE_SWITCH_MODE_GLOBAL means that the LSS inquiry is to be initialized with a SwitchModeGlobal command
VendorId	(in)	Vendor ID of the device to be inquired
ProductCode	(in)	Product code of the device to be inquired
RevisionNo	(in)	Revision number of the device to be inquired
SerialNo	(in)	Serial number of the device to be inquired
node_no	(out)	Node-ID of the device to be inquired (between 1 and 127, possibly as special value 255)

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Order could not be entered in transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Success
COP_k_NO	General error of the Master Firmware
COP_k_TIMEOUT	No response from device
LSS_k_MEDIA_ACCESS_ERROR	CAN bus access failed
LSS_k_IV_PARAMETER	Unauthorized parameter value
LSS_k_PROTOCOL_ERR	Unauthorized device response (LSS protocol violation)
LSS_k_BSY	LSS module of the firmware already busy

5.6.4 COP_LSS_ConfigNodeID

Description: COP_LSS_ConfigNodeID sets the node-ID of a CANopen device using the LSS services.
A special value of 255 for the node-ID means that the device is to be set to the so-called "LSS Init State".
In this function a complete block of LSS commands is transmitted in the following order:

(1a) SwitchModeGlobal to activate the LSS service, if defined.

(1b) Alternatively SwitchModeSelective to address the device using the parameters VendorId, ProductCode, RevisionNo and SerialNo.

(2) ConfigNodeID to configure the node-ID using the parameter new_node_no.

(3) StoreConfiguration to store the configuration, if defined.

(4) SwitchModeGlobal to deactivate the LSS service.

If SwitchModeGlobal is used with the relevant flag, the CANopen API function can only be used in cases where only *one* LSS-compatible device exists in the network, as otherwise the device responses are superimposed and disturb each other.

Prototype:

```
short COP_LSS_ConfigNodeID(
                        COP_t_HANDLE boardhdl,
                        BYTE         baudtable,
                        BYTE         baudrate,
                        BYTE         mode,
                        DWORD        VendorId,
                        DWORD        ProductCode,
                        DWORD        RevisionNo,
                        DWORD        SerialNo,
                        BYTE         new_node_no );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
baudtable	(in)	Selection of the bittiming table to be used for the CAN communication during the LSS configuration process. There are two different tables:: COP_k_BAUD_CIA Table with the bittimings specified by CiA in CiA-301 Standard table. COP_k_BAUD_USER Table with userdefined bittiming settings. These must have been set before by a further API function.

baudrate	(in)	Baudrate for the CAN communication during the LSS configuration process. However, does **not** overwrite the baudrate possibly configured with COP_InitInterface() . The following values are permitted: COP_k_10_KB COP_k_20_KB COP_k_50_KB COP_k_100_KB COP_k_125_KB COP_k_250_KB COP_k_500_KB COP_k_800_KB COP_k_1000_KB
mode	(in)	Operating mode of the function. Possible values are: LSS_k_SET_MODE_SWITCH_MODE_GLOBAL means that the LSS inquiry is to be initialized with a SwitchModeGlobal command LSS_k_SET_MODE_STORE_CONFIGURATION means that the LSS command to save the new node-ID is to be transmitted
VendorId	(in)	Vendor-ID of the addressed device
ProductCode	(in)	Product-code of the addressed device
RevisionNo	(in)	Revision-number of the addressed device
SerialNo	(in)	Serial-number of the addressed device
new_node_no	(in)	New node-ID of the addressed device (between 1 and 127, possibly a special value 255)

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Order could not be entered in transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Success
COP_k_NO	General error of the Master Firmware
COP_k_TIMEOUT	No response from device
LSS_k_MEDIA_ACCESS_ERROR	CAN bus access failed
LSS_k_IV_PARAMETER	Unauthorized parameter value
LSS_k_PROTOCOL_ERR	Unauthorized device response (LSS protocol violation)
LSS_k_BSY	LSS module of the firmware already busy

5.6.5 COP_LSS_ConfigBitTiming

Description: `COP_LSS_ConfigBitTiming` configures the baudrate of a CANopen device using the LSS services.
In this function a complete block of LSS commands is transmitted in the following order:

(1a) SwitchModeGlobal to activate the LSS service, if defined.

(1b) Alternatively SwitchModeSelective to address the device using the parameters `VendorId`, `ProductCode`, `RevisionNo` and `SerialNo`.

(2) ConfigureBitTimingParameters() to configure the baudrate using the parameters `new_bautable` and `new_baudrate`.

(3) ActivateBitTimingParameters to directly activate the new baudrate after a configurable delay time using the parameter `switch_delay`.

(4) StoreConfiguration to store the configuration, if defined.

(5) SwitchModeGlobal to deactivate the LSS service.

If SwitchModeGlobal is used with the relevant flag, the CANopen API function can only be used in cases where only *one* LSS-compatible device exists in the network, as otherwise the device responses are superimposed and disturb each other.

When this COP_LSS_ConfigBitTiming() function is called, the firmware is reset internally. Then a complete re-initialization of the CANopen Master Firmware is therefore necessary in accordance with Fig. 2-1, beginning with COP_InitInterface().

Prototype:

```
short COP_LSS_ConfigBitTiming(
                              COP_t_HANDLE boardhdl,
                              BYTE         baudtable,
                              BYTE         baudrate,
                              BYTE         mode,
                              DWORD        VendorId,
                              DWORD        ProductCode,
                              DWORD        RevisionNo,
                              DWORD        SerialNo,
                              BYTE         new_baudtable,
                              BYTE         new_baudrate,
                              WORD         switch_delay );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
baudtable	(in)	Selection of the bittiming table to be used for the CAN communication during the LSS configuration process. There are two different tables:: COP_k_BAUD_CIA Table with the bittimings specified by CiA in CiA-301 Standard table. COP_k_BAUD_USER Table with userdefined bittiming settings. These must have been set before by a further API function.
baudrate	(in)	Baudrate for the CAN communication during the LSS configuration process. However, does **not** overwrite the baudrate possibly already configured with COP_InitInterface() . The following values are permitted: COP_k_10_KB COP_k_20_KB COP_k_50_KB COP_k_100_KB COP_k_125_KB COP_k_250_KB COP_k_500_KB COP_k_800_KB COP_k_1000_KB
mode	(in)	Operating mode of the function. Possible values are: LSS_k_SET_MODE_SWITCH_MODE_GLOBAL means that the LSS inquiry is to be initialized with a SwitchModeGlobal command LSS_k_SET_MODE_ACTIVATE_NEW_BAUDRATE means that the LSS command to activate the new baudrate is to be transmitted after expiry of the specified delay time configured with switch_delay. LSS_k_SET_MODE_STORE_CONFIGURATION means that the LSS command to store the new baudrate is to be transmitted
VendorId	(in)	Vendor-ID of the addressed device
ProductCode	(in)	Product-code of the addressed device
RevisionNo	(in)	Revision-number of the addressed device
SerialNo	(in)	Serial-number of the addressed device
new_baudtable	(in)	New bittiming table of the addressed device. Value COP_k_BAUD_CIA is the standard setting, however, there are manufacturer specific values permitted above 127:

new_baudrate	(in)	New baudrate of the addressed device The following values are permitted: COP_k_10_KB COP_k_20_KB COP_k_50_KB COP_k_100_KB COP_k_125_KB COP_k_250_KB COP_k_500_KB COP_k_800_KB COP_k_1000_KB
switch_delay	(in)	Delay time before possible activation of the new baudrate in milliseconds.

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Order could not be entered in transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Success
COP_k_NO	General error of the Master Firmware
COP_k_TIMEOUT	No response from device
LSS_k_MEDIA_ACCESS_ERROR	CAN bus access failed
LSS_k_IV_PARAMETER	Unauthorized parameter value
LSS_k_PROTOCOL_ERR	Unauthorized device response (LSS protocol violation)
LSS_k_BSY	LSS module of the firmware already busy

5.6.6 COP_LSS_ActivateBitTiming

Description: COP_LSS_ActivateBitTiming is used to switch the baudrate of all connected CANopen devices at the same time using the LSS service.
In this function a complete block of LSS commands is transmitted in the following order:

(1) SwitchModeGlobal to activate the LSS service.
(2) ActivateBitTimingParameters to activate the new baudrate after a configurable delay time using the parameters switch_delay.
(3) SwitchModeGlobal to activate the LSS service.

When this COP_LSS_ActivateBitTiming function is called, the firmware is reset internally. Then a complete re-initialization of the CANopen Master Firmware is therefore necessary in accordance with Fig. 2-1, beginning with COP_InitInterface().

Prototype:

```
short COP_LSS_ActivateBitTiming(
                        COP_t_HANDLE boardhdl,
                        BYTE         baudtable,
                        BYTE         baudrate,
                        BYTE         new_baudtable,
                        BYTE         new_baudrate,
                        WORD         switch_delay );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN-board/line combination
baudtable	(in)	Selection of the bittiming table to be used for the CAN communication during the LSS configuration process. There are two different tables:: COP_k_BAUD_CIA Table with the bittimings specified by CiA in CiA-301 Standard table. COP_k_BAUD_USER Table with userdefined bittiming settings. These must have been set before by a further API function.

baudrate	(in)	Baudrate for the CAN communication during the LSS configuration process. However, does **not** overwrite the baudrate possibly configured with COP_InitInterface() . The following values are permitted: COP_k_10_KB COP_k_20_KB COP_k_50_KB COP_k_100_KB COP_k_125_KB COP_k_250_KB COP_k_500_KB COP_k_800_KB COP_k_1000_KB
new_baudtable	(in)	New bittiming table of the Master Firmware. Permissible values are COP_k_BAUD_CIA and COP_k_BAUD_USER.
new_baudrate	(in)	New baudrate of the Master Firmware. The following values are permitted: COP_k_10_KB COP_k_20_KB COP_k_50_KB COP_k_100_KB COP_k_125_KB COP_k_250_KB COP_k_500_KB COP_k_800_KB COP_k_1000_KB This parameter does not replace the configuration of the baudrate of each individual node COP_LSS_ConfigBitTiming, but informs the firmwareto which baudrate it should set itself after switching the network baudrate.
switch_delay	(in)	Delay time before activating the baudrate in milliseconds.

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Order could not be entered in transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Success
COP_k_NO	General error of the Master Firmware
COP_k_TIMEOUT	No response from device
LSS_k_MEDIA_ACCESS_ERROR	CAN bus access failed
LSS_k_IV_PARAMETER	Unauthorized parameter value
LSS_k_PROTOCOL_ERR	Unauthorized device response (LSS protocol violation)
LSS_k_BSY	LSS module of the firmware already busy

5.6.7 COP_LSS_IdentifyRemoteSlaves

Description: With COP_LSS_IdentifyRemoteSlaves, devices of which the vendor-ID and the product-code are known can be identified in the network.
If more than one device of the same type exists in the network, or if the serial numbers or revision numbers are not known, an individual CANopen device that supports this LSS service can be clearly identified with this function through interactive narrowing of the serial number path.
This function does not require isolated CAN communication and only transmits one LSS command:

(1) LSSIdentifyRemoteSlave using the parameters VendorId, ProductCode, RevisionNoLow, RevisionNoHigh, SerialNoLow and SerialNoHigh.

If at least one node responds from the specified serial number path, the function returns COP_k_OK, otherwise COP_k_TIMEOUT. The delay time until the device response can be configured with the function COP_SetLSSTimeOut().

Prototype:

```
short COP_LSS_IdentifyRemoteSlaves(
                              COP_t_HANDLE boardhdl,
                              BYTE         baudtable,
                              BYTE         baudrate,
                              DWORD        VendorId,
                              DWORD        ProductCode,
                              DWORD        RevisionNoLow,
                              DWORD        RevisionNoHigh,
                              DWORD        SerialNoLow,
                              DWORD        SerialNoHigh );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN board/line combination
baudtable	(in)	Selection of the bittiming table to be used for the CAN communication during the LSS configuration process. There are two different tables:: COP_k_BAUD_CIA Table with the bittimings specified by CiA in CiA-301 Standard table. COP_k_BAUD_USER Table with userdefined bittiming settings. These must have been set before by a further API function.

baudrate	(in)	Baudrate for the CAN communication during the LSS configuration process. However, does **not** overwrite the baudrate possibly configured with COP_InitInterface() The following values are permitted: COP_k_10_KB COP_k_20_KB COP_k_50_KB COP_k_100_KB COP_k_125_KB COP_k_250_KB COP_k_500_KB COP_k_800_KB COP_k_1000_KB
VendorId	(in)	Vendor-ID of the devices to be identified
ProductCode	(in)	Product-code of the devices to be identified
RevisionNoLow	(in)	Lower revision-number limit of the devices to be identified
RevisionNoHighw	(in)	Upper revision-number limit of the devices to be identified
SerialNoLow	(in)	Lower serial-number limit of the devices to be identified
SerialNoHigh	(in)	Upper serial-number limit of the devices to be identified

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Order could not be entered in transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Success
COP_k_NO	General error of the Master Firmware
COP_k_TIMEOUT	No response from device
LSS_k_MEDIA_ACCESS_ERROR	CAN bus access failed
LSS_k_IV_PARAMETER	Unauthorized parameter value
LSS_k_PROTOCOL_ERR	Unauthorized device response (LSS protocol violation)
LSS_k_BSY	LSS module of the firmware already busy

5.6.8 COP_LSS_IdentifyNonConfRemoteSlaves

Description: COP_LSS_IdentifyRemoteSlaves identifies whether devices exist in the network in the so-called "LSS Waiting State".
This function does not require isolated CAN communication and only transmits one LSS command:
(1) LSSIdentifyNonConfiguredRemoteSlaves.
If at least one node responds from the specified serial number path, the function returns COP_k_OK, otherwise COP_k_TIMEOUT. The delay time until the device response can be configured with the function COP_SetLSSTimeOut().

Prototype:

```
short COP_LSS_IdentifyNonConfRemoteSlaves(
                                COP_t_HANDLE boardhdl,
                                BYTE         baudtable,
                                BYTE         baudrate );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN board/line combination
baudtable	(in)	Selection of the bittiming table to be used for the CAN communication during the LSS configuration process. There are two different tables: COP_k_BAUD_CIA Table with the bittimings specified by CiA in CiA-301 Standard table. COP_k_BAUD_USER Table with userdefined bittiming settings. These must have been set before by a further API function.
baudrate	(in)	Baudrate for the CAN communication during the LSS configuration process. However, does **not** overwrite the baudrate possibly configured with COP_InitInterface() The following values are permitted: COP_k_10_KB COP_k_20_KB COP_k_50_KB COP_k_100_KB COP_k_125_KB COP_k_250_KB COP_k_500_KB COP_k_800_KB COP_k_1000_KB

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Order could not be entered in transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Success
COP_k_NO	General error of the Master Firmware
COP_k_TIMEOUT	No response from device
LSS_k_MEDIA_ACCESS_ERROR	CAN bus access failed
LSS_k_IV_PARAMETER	Unauthorized parameter value
LSS_k_PROTOCOL_ERR	Unauthorized device response (LSS protocol violation)
LSS_k_BSY	LSS module of the firmware already busy

5.6.9 COP_LSS_Fastscan

Description: With COP_LSS_Fastscan a device in the network in the so-called "LSS Waiting State" will be found.
This function does not require isolated CAN communication and also does not need a vendor-id or a product-code specified, for the entire LSS address range will be searched for systemtically.
In case there are several devices of the same model present in the network, this function finds exactly one of them, by iterative and bitwise narrowing of the LSS address, and finally delivers its precise vendor-id, product-code, revision-number and serial-number. By using these device identification data the particular device shall be configured subsequently using the other LSS services functions provided such as COP_LSS_ConfigNodeID().
By followup calls of this function another device will be found and can be configured, until finally all unconfigured network participants have been detected. If a node has been found, the function returns COP_k_OK, otherwise LSS_k_FS_NO_NONCONFIGURED_SLAVE or LSS_k_FS_NF_NONCONFIGURED_SLAVE respectively. The delay time until the device response can be configured with the function COP_SetLSSTimeOut().
Due to the multitude of LSS commands sent on the bus during a fastscan run, and the LSS timeout interval being applied to each one of it, the overall time of the fastscan process is up to 130-times(!) the simple LSS timeout value, and so it sums up to 13 seconds when using the default LSS timeout value of 100ms. For that amount of time the function will not return to the calling application, hence the LSS timeout value should be reduced notably prior to executing a LSS fastscan, to 40ms, for instance, resulting in an overall execution time of 5 seconds for a single LSS fastscan run.

Prototype:

```
short COP_LSS_Fastscan(
                        COP_t_HANDLE boardhdl,
                        BYTE         baudtable,
                        BYTE         baudrate,
                        DWORD*       VendorId,
                        BYTE         VendorIdBits,
                        DWORD*       ProductCode,
                        BYTE         ProductCodeBits,
                        DWORD*       RevisionNo,
                        BYTE         RevisionNoBits,
                        DWORD*       SerialNo,
                        BYTE         SerialNoBits );
```

Parameters:

Parameter	Dir.	Explanation
boardhdl	(in)	Handle of the CAN board/line combination
baudtable	(in)	Selection of the bittiming table to be used for the CAN communication during the LSS scan process. There are two different tables:: COP_k_BAUD_CIA Table with the bittimings specified by CiA in CiA-301 Standard table. COP_k_BAUD_USER Table with userdefined bittiming settings. These must have been set before by a further API function.
baudrate	(in)	Baudrate for the CAN communication during the LSS scan process. However, does **not** overwrite the baudrate possibly configured with COP_InitInterface() . The following values are permitted: COP_k_10_KB COP_k_20_KB COP_k_50_KB COP_k_100_KB COP_k_125_KB COP_k_250_KB COP_k_500_KB COP_k_800_KB COP_k_1000_KB
VendorId	(in/out)	Preset resp detected vendor identification. As input parameter: preset value of the vendor identification to shorten the scan process; Otherwise 0 to find any device. As output parameter: vendor identification of the found device.
VendorIdBits	(in)	Number of bits to be checked of given vendor identification starting left at the MSB (Value between 0 and 31) For a full range scan and with no preset vendor identification value in VendorId argument, this value has to be set to 31 rather than 0!
ProductCode	(in/out)	Preset resp detected product-code. As input parameter: preset value of the product to shorten the scan process; Otherwise 0 to find any device. As output parameter: product-code of the found device.
ProductCodeBits	(in)	Number of bits to be checked of given product-code starting left at the MSB (Value between 0 and 31) For a full range scan and with no preset product-code value in ProductCode argument, this value has to be set to 31 rather than 0!

RevisionNo	(in/out)	Preset resp detected product revision-number. As input parameter: preset value of the revision-number to shorten the scan process; Otherwise 0 to find any device. As output parameter: product revision-number of the found device.
RevisionNoBits	(in)	Number of bits to be checked of given revision-number starting left at the MSB (Value between 0 and 31) For a full range scan and with no preset revision-number value in RevisionNo argument, this value has to be set to 31 rather than 0!
SerialNo	(in/out)	Preset resp detected device serial-number. As input parameter: preset value of the serial-number to shorten the scan process; Otherwise 0 to find any device. As output parameter: serial-number of the found device.
RevisionNoBits	(in)	Number of bits to be checked of given device serial-number starting left at the MSB (Value between 0 and 31) For a full range scan and with no preset serial number value in SerialNo argument, this value has to be set to 31 rather than 0!

Return values:

Return value	Description
BER_k_ERR	Handle invalid
BER_k_NOT_SENT	Order could not be entered in transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Success
COP_k_NO	General error of the Master Firmware
LSS_k_MEDIA_ACCESS_ERROR	CAN bus access failed
LSS_k_IV_PARAMETER	Unauthorized parameter value
LSS_k_PROTOCOL_ERR	Unauthorized device response (LSS protocol violation)
LSS_k_BSY	LSS module of the firmware already busy
LSS_k_FS_NO_NONCONFIGURED_SLAVE	No response, nothing found
LSS_k_FS_NO_NONCONFIGURED_SLAVE	No response from device, preset device not found

Appendix A - Error codes

The error codes of the CANopen Master API DLL

These error codes come from the CANopen Master API DLL and signal errors in the communication between the Master Firmware and the API DLL as well as internal problems of the API DLL. The explanations of all error codes used from the function descriptions are summarized again here and supplemented by debugging tips.

Error code name **Error code value**	**Description**
BER_k_OK 0	Success
BER_k_ERR 1	General error, not further specified Is generally returned when the specified Boardhandle is invalid, but can also be used in other error situations.
BER_k_DATA_CORRUPT -41	Incorrect data patterns received. This indicates a corruption of the communication path from firmware to windows library (internal error). Try to repeat the function call.
BER_k_NOT_SENT -40	Congestion of the communication path from windowd library to the firmware (internal error) Try to repeat the function call after a while.
BER_k_NO_NEW_MSG -39	- not used -
BER_k_TIMEOUT -38	Communication timeout. This means that the firmware has not responded within the communication delay time (CommTimeOut) (internal error) Most likely, the VCI communication channel is broken.
BER_k_BOARD_ALREADY_USED -37	Required CAN-board is already being used by CANopen Master API A frequent source of error is the use of an interpreter language. If aborted during debugging and the board is therefore not deregistered correctly, the API no longer releases the board for a 'further' process. Try to reset the complete DLL with `COP_Reset_DLL()`. (Attention: only during the development phase!) This is a fatal error stopping Master API operation.
BER_k_ALL_BOARDS_USED -36	Master API has reached maxium capacity of 12 simultaneously manageable CAN-boards. See previous error code. This is a fatal error stopping Master API operation.
BER_k_BOARD_NOT_SUPP -35	Requested CAN-board is not supported by CANopen Master API due to unsuitable microcontroller or memory extension Use generic mode `COP_VCI3GENERIC` instead.

BER_k_BOARD_NOT_FOUND -34	Specified board type and identification do not match any available CAN-board This is a fatal error stopping Master API operation.
BER_k_CANNOT_SEARCH_BOARD -33	User has aborted the hardware selection dialog with "Cancel"
BER_k_WRONG_FW -32	The version number supplied by the firmware is incorrect (internal error). Indicates malfunctioning communication between PC and microcontroller This is a fatal error stopping Master API operation.
BER_k_USED_FROM_OTHER_PROCESS -31	Requested CAN-board is already occupied by another CAN-application This is a fatal error stopping Master API operation.
BER_k_PC_MC_COMM_ERR -30	Communication link with the CAN-board is disturbed
BER_k_BOARD_DLD_ERR -29	An error has occured during firmware download. This normally indicates a VCI installation problem. Please read the installation instructions of the VCI or contact the technical support. It also results from a missing generic firmware library file XatCOP60_VCI3.dll This is a fatal error stopping Master API operation.
BER_k_BADCALLBACK_PTR -28	An invalid function pointer was given
BER_k_NO_SUCH_CANLINE -27	Requested CAN line is not installed or is not supported by the firmware The second CAN line cannot be used on CAN boards with an 8-bit microcontroller even though it may be installed, as the power of the processor is insufficient. The same applies when the single line firmware is selected via the COP_InitBoard() argument lCANline = COP_SINGLELINE
BER_k_CANLINE_USED -26	Requested CAN line is already busy COP_InitBoard() has probably already been called for this CAN line. The CAN lines can nevertheless be used dynamically, i.e. it is possible to release a CAN line used in a running program again with COP_ReleaseBoard(), as well as initialize another one with COP_InitBoard().
BER_k_VCI_INST_ERR -25	IXXAT basic driver VCI not available, incomplete or otherwise defective In addition to an installed CAN board, the condition for the CANopen Master API is correct installation of the board driver via the VCI set-up program. See also section 2.3 Before installation. This is a fatal error stopping Master API operation.
BER_k_BOARD_ERR -24	Incorrect or unknown board type. This is a fatal error stopping Master API operation.
BER_k_MEM_ALLOC_ERR -23	Internal data structures or operating system objects could not be created. This is a fatal error stopping Master API operation.

BER_k_CCI_INST_ERR -22	CCI installation error (internal) Please make a note of the Hardware version number from your CAN-board's nameplate and report it to IXXAT support
BER_k_SDO_INST_ERR -21	Internal error when instancing or configuring the SDO Thread. This is a fatal error stopping Master API operation.
BER_k_SDO_THREAD_ERR -20	Error in the control of the SDO-Thread or SDO-Thread cancelled by COP_CancelSDO(). Try to repeat the function call.

The error codes of the CANopen Master Firmware

These error codes come from the Master Firmware and signal errors in the communication between the CAN-Controller and the Master Firmware as well as internal problems of the firmware. Here the explanations of the individual values from the function descriptions are summarized again.

Error code name **Error code value**	**Description**
COP_k_OK 0	Success
COP_k_NO 1	General error, not further specified
COP_k_CAL_ERR 2	General error of the CANopen Master Firmware
COP_k_IV 3	Invalid parameter value
COP_k_ABORT 4	SDO-Transfer aborted
COP_k_NOT_FOUND 5	No node registered with the stated node-ID Indicates that the call of COP_AddNode() was forgotten for this node-ID.
COP_k_NOT_INIT 6	Master has not yet been initialized by means of COP_InitInterface()
COP_k_INIT 7	Master is initialized and ready for use This is no error, but a valid regular return code.
COP_k_NO_OBJECTS COP_k_QUEUE_EMPTY 9	No new entries available in data queue This is no error, but a valid regular return code.
COP_k_TIMEOUT 10	No response from the device
COP_k_SDO_RUNNING 16	SDO-Transfer in progress Wait some time, then retry.
COP_k_BSY 17	CAN transmit-queue is full or firmware is otherwise engaged Wait a few milliseconds, then retry.
COP_k_NO_OBJECT 18	Local object dictionary entry is not existing (internal error). Used only with SDO-Manager Feature.
COP_k_NO_SUBINDEX 19	Local object dictionary entry is not existing (internal error). Used only with SDO-Manager Feature.

COP_k_PRESENT_DEVICE_STATE 21	Access currently not possible Used only with SDO-Manager Feature.
COP_k_RANGE_EXCEEDED 22	Parameter out of range Used only with SDO-Manager Feature.
COP_k_UNKNOWN 32	Unknown Opcode
COP_k_NO_FLY_MASTER_PRESENT 33	Flying Master functionality not supported or not activated
COP_k_NO_LOWSPEED 34	LowSpeed bus coupling is not present or is not supported for the particular CAN board

Appendix B - Performance characteristics

The following table lists some of the capacity limit values for the supportedCAN-boards. These values are defined by the microcontroller and the memory extension of the CAN board and by the implementation of the CANopen Master Firmware itself.

	iPC-I 320 tinCAN V4 USB-to-CAN compact	**Generic VCI3 e.g. USB-to-CAN V2**	**CAN-IB200 iPC-I XC16 tinCAN161 USB-to-CAN II CAN@net II**
Maximum number of simul-taneously manageable nodes (COP_AddNode)	40 + 1	127 + 1	
Maximum number of TPDOs per node (COP_CreatePDO)	12	16	
Maximum number of RPDOs per node (COP_CreatePDO)	12	16	
Maximum number of SDOs per node (COP_CreateSDO)	2 (incl. default-SDO)		
Maximum number of parallel running SDO-Transfers	1		
Maximum total number of RPDOs and TPDOs	500	1600[1]	
Maximum number of synchronous RPDOs and TPDOs	200	600	
Multi line operation	No	Yes[2]	
Flying Master additional functionality	No	Yes	

[1] The maximum number of PDOs according to Predefined Connection Set defined in the CANopen specification /1/ is 1016

[2] No for CAN@netII because of a single existing CAN controller

Appendix C - Scope of delivery

The CANopen Master API is installed as standard in the directory `\Programs\IXXAT\CANopen Master API 6.1`

The sample applications are located at `\Users\Public\Documents\IXXAT CANopen Master API 6.1`

Neither the program library nor the demo applications enter program settings in the Windows registry.

Folder	File name	Meaning
\		Documentation
	files.txt	List of installed files and their version numbers
	LiesMich.txt	Important last-minute product information (German)
	ReadMe.txt	Important last-minute product information (English)
	HISTORY.txt	Development history and record of changes
	4.12.0132.10000.pdf	Electronic operating manual and documentation (German)
	4.12.0132.20000.pdf	Electronic operating manual and documentation (English)

\bin		Binary files
	XatCOP60.dll	CANopen Master API 6
	XatCOP60-64.dll	CANopen Master API 6 (64bit version for processor architecture "amd64")
	XatCOP_VCI3.dll	Generic VCI3 firmware
	XatCOP_VCI3-64.dll	Generic VCI3 firmware (64bit version for processor architecture "amd64")
	MdllWrapper6.dll	COM-Wrapper of the CANopen Master API 6
\bin\debuglog		Debug-version (generates protocol file `XatCOP60.LOG`)
	XatCOP60.dll	CANopen Master API 6
	XatCOP60-64.dll	CANopen Master API 6 (64bit version for processor architecture "amd64")

\lib		Library files for the C Compilers
\lib\BCB		Lib-file for the Borland C++ Builder
	XatCOP60.lib	Lib-file for C++Builder in OMF format, generated with the command line tool `implib.exe`
	Readme.txt	Instructions for generating the Lib-file
\lib\MSVC		Lib-files for Microsoft Visual C++ / Visual Studio
	XatCOP60.lib	Lib-file for MS Visual C / Visual Studio
	XatCOP60-64.lib	Lib-file for MS Visual C / Visual Studio (64bit version)

\Samples\		Headers for all programming languages with constants, types and interface descriptions
	cop.bas	Visual Basic (legacyt) main header of the CANopen Master API
	cop.h	C main header of the CANopen Master API
	cop.pas	Delphi (Pascal) main header of the CANopen Master API
	cop.cs	C# Main header of the CANopen Master API
	cop.vb	Visual Basic.NET Main header of the CANopen Master API
	copcmd.h	C header with the command-Opcodes of the CANopen Master API
	copcmd.pas	Delphi (Pascal) header with the command-Opcodes of the CANopen Master API
	CopSDOManager.h	C header containing additional CANopen Master API functions
	CopUserBittiming.h	C header containing additional CANopen Master API functions
	LSScmd.h	C header with the LSS constants
	CopError.h	C header for issuing description texts for the return error codes
	CopError.c	Corresponding C implementation file
	XatBrds.bas	All IXXAT board types (CAN boards) for Visual Basic (legacy)
	XatBrds.h	C header of all IXXAT board types (CAN boards VCI 2)
	XatBrds.pas	Delphi (Pascal) header of all IXXAT board types (CAN boards VCI 2)
	XatBrds.cs	All IXXAT board types (CAN boards) for C#
	XatBrds.vb	All IXXAT board types (CAN boards) for Visual Basic .NET
	Vciguid.h	C header of all IXXAT board types (CAN boards VCI3)
	Vci3Guid.pas	Delphi (pascal) header of all IXXAT board types (CAN boards VCI3)

\Samples\C		C sample applications
\Samples\C\BCB DigIO		Borland C++ Builder sample application for control of an external DS-401 device via its two default-PDOs of length 1
	DigIOdemo.bpr	BCB project file
	DigIOdemo.cpp	Implementation of WinMain()
	DigIOdemo.exe	Release-Version of the sample application
	DigIOdemo.exe.manifest	Declaration of the common controls utilised for support of XP visual styles
	XatCOP60.dll	CANopen Master API 6
	XatCOP_VCI3.dll	Generic VCI3 firmware
	DigIOdemo.ico	Icon of the application
	DigIOdemo.res	Binary resource file, contains the application-icon
	Main.cpp	Implementation of the main window of the application
	Main.dfm	Configuration of the control elements in the main window of the application
	Main.h	Corresponding header of the main window

\Samples\C\BCB NetManage		Borland C++ Builder sample application for the object dictionary access to external devices via the default SDO
	FileOpen.bmp	Bitmap for button
	FileSave.bmp	Bitmap for button
	Main.dfm	Configuration of the control elements in the main window of the application
	Main.cpp	Implementation of the main window of the application
	Main.h	Corresponding header of the main window
	NetManage.bpr	BCB project file
	NetManage.cpp	Implementation of WinMain()
	NetManage.exe	Release-Version of the sample application
	NetManage.exe.manifest	Declaration of the common controls utilised for support of XP visual styles
	XatCOP60.dll	CANopen Master API 6
	XatCOP60_VCI3.dll	Generic VCI3 firmware
	NetManage.res	Binary resource file, contains the application-icon
	NetManage.ico	Icon of the application

\Samples\C\MFC DigIO		MFC sample application for control of an external DS-401 device via its two default-PDOs of length 1 for Visual Studio 2005 and Visual Studio 6
	DigIOdemo.cpp	Implementation of the application class
	DigIOdemo.dsp	Visual Studio 6 project file
	DigIOdemo.exe	Release build of the sample application
	DigIOdemo.exe.manifest	Declaration of the common controls utilised for support of XP visual styles
	XatCOP60.dll	CANopen Master API 6
	XatCOP_VCI3.dll	Generic VCI3 firmware
	DigIOdemo.h	Header of the application class
	DigIOdemo.rc	Resource script of the application
	DigIOdemo.sln	Visual Studio 2005 project file
	DigIOdemo.vcproj	Visual C++ 2005 project file
	DigIOdemoDlg.cpp	Implementation of the main window of the application
	DigIOdemoDlg.h	Header of the main window of the application
	ReadMe.txt	Info file generated by MFC Class Wizard
	Resource.h	Resource-IDs of the control elements
	StateLED.cpp	Implementation of a round colored status LED **CStaticLED** derived from **CStatic**
	StateLED.h	Corresponding header
	StdAfx.cpp	Configuration of the MFC elements used
	StdAfx.h	Header for configuration of the MFC elements used
\Samples\C\MFC DigIO\res		Resource files
	DigIOdemo.ico	Icon of the application
	DigIOdemo.rc2	Visual C++ 6 additional resources
	IXATLOGO.BMP	Image file
	LEDclear.bmp	Image file of the gray status LED
	LEDin.bmp	Image file of the green status LED
	LEDout.bmp	Image file of the red status LED
\Samples\C\MFC DigIO\x64		64bit version of the sample application for processor architecture “amd64”
	DigIOdemo.exe	Release build of the sample application
	DigIOdemo.exe.manifest	Declaration of the common controls utilised for support of XP visual styles
	XatCOP60.dll	CANopen Master API 6 (64bit version)
	XatCOP_VCI3-64.dll	Generic VCI3 firmware (64bit version)

\Samples\C\Win32 App		Sample application for monitoring of an external DS-401 device and control via its default RPDO1 of length 2 developed without class library with the pure Windows API for Visual Studio 2005 and Visual Studio 6
	COPMSMPL.cpp	Implementation of the application and of the application window
	COPMSMPL.h	Corresponding header
	COPMSMPL.dsp	Visual Studio 6 project file
	COPMSMPL.exe	Release build of the sample application
	COPMSMPL.exe.manifest	Declaration of the Common Controls version used for XP visual styles
	XatCOP60.dll	CANopen Master API 6
	XatCOP_VCI3.dll	Generic VCI3 firmware
	COPMSMPL.rc	Resource script of the application
	resource.h	Resource script of the application
	COPMSMPL.sln	Visual Studio 2005 project file
	COPMSMPL.vcproj	Visual C++ 2005 project file
\Samples\C\Win32 App\x64		64bit version of the sample application for processor architecture "amd64"
	COPMSMPL.exe	Release build of the sample application
	COPMSMPL.exe.manifest	Declaration of the Common Controls version used for XP visual styles
	XatCOP60-64.dll	CANopen Master API 6 (64bit version)
	XatCOP_VCI3-64.dll	Generic VCI3 firmware (64bit version)

\Samples\C\VS2010 MFC DigIO		MFC sample application for control of an external DS-401 device via its two default-PDOs of length 1 for Visual Studio 2010
	DigIOdemo.cpp	Implementation of the application class
	DigIOdemo.exe	Release build of the sample application
	DigIOdemo.exe.manifest	Declaration of the common controls utilised for support of XP visual styles
	XatCOP60.dll	CANopen Master API 6
	XatCOP_VCI3.dll	Generic VCI3 firmware
	DigIOdemo.h	Header of the application class
	DigIOdemo.rc	Resource script of the application
	DigIOdemo.sln	Visual Studio 2010 project file
	DigIOdemo.vcxproj	Visual C++ 2010 project file
	DigIOdemo.vcxproj.filter	Visual C++ 2010 project file
	DigIOdemoDlg.cpp	Implementation of the main window of the application
	DigIOdemoDlg.h	Header of the main window of the application
	ReadMe.txt	Info file generated by MFC Class Wizard
	Resource.h	Resource-IDs of the control elements
	StateLED.cpp	Implementation of a round colored status LED **CStaticLED** derived from **CStatic**
	StateLED.h	Corresponding header
	StdAfx.cpp	Configuration of the MFC elements used

	StdAfx.h	Header for configuration of the MFC elements used
\Samples\C\VS2010 MFC DigIO\res		Resource files
	DigIOdemo.ico	Icon of the application
	DigIOdemo.rc2	Additional resources
	IXATLOGO.BMP	Image file
	LEDclear.bmp	Image file of the gray status LED
	LEDin.bmp	Image file of the green status LED
	LEDout.bmp	Image file of the red status LED
\Samples\C\VS2010 MFC DigIO\x64		64bit version of the sample application for processor architecture "amd64"
	DigIOdemo.exe	Release build of the sample application
	DigIOdemo.exe.manifest	Declaration of the common controls utilised for support of XP visual styles
	XatCOP60.dll	CANopen Master API 6 (64bit version)
	XatCOP_VCI3-64.dll	Generic VCI3 firmware (64bit version)

\Samples\C\VS2010 Win32 App		Sample application for monitoring of an external DS-401 device and control via its default RPDO1 of length 2 developed without class library with the pure Windows API for Visual Studio 2010
	COPMSMPL.cpp	Implementation of the application and of the application window
	COPMSMPL.h	Corresponding header
	COPMSMPL.exe	Release build of the sample application
	COPMSMPL.exe.manifest	Declaration of the Common Controls version used for XP visual styles
	XatCOP60.dll	CANopen Master API 6
	XatCOP_VCI3.dll	Generic VCI3 firmware
	COPMSMPL.rc	Resource script of the application
	resource.h	Resource script of the application
	COPMSMPL.sln	Visual Studio 2010 project file
	COPMSMPL.vcxproj	Visual C++ 2010 project file
	COPMSMPL.vcxproj.filter	Visual C++ 2010 project file
\Samples\C\VS2010 Win32 App\x64		64bit version of the sample application for processor architecture "amd64"
	COPMSMPL.exe	Release build of the sample application
	COPMSMPL.exe.manifest	Declaration of the Common Controls version used for XP visual styles
	XatCOP60-64.dll	CANopen Master API 6 (64bit version)
	XatCOP_VCI3-64.dll	Generic VCI3 firmware (64bit version)

\Samples\C#		C# Sample applications
\Samples\C#\DigIO		C# Sample application to control an external DS-401 device via its two default PDOs of length 1 for Visual Studio 2008 and .NET 2.0 Framework
	cop.cs	Copy of the CANopen Master API C# main header
	DigIOdemo.csproj	Visual C# 2008 project file
	DigIOdemo.exe	Release build of the sample application
	XatCOP60.dll	CANopen Master API 6
	XatCOP_VCI3.dll	Generic VCI3 firmware
	DigIOdemo.ico	Icon of the application
	DigIOdemo.sln	Visual Studio 2008 project file
	DigIOdemo.csproj.user	Visual C# 2008 project settings
	MainForm.cs	Implementation of the main window of the application
	MainForm.resx	Configuration of the control elements in the main window of the application
	LED.cs	Implementation of a round colored status LED as a user-defined control element
	LED.resx	Configuration of the control elements for user-defined coloured status LED
	XatBrds.cs	Copy of the IXXAT board type file (CAN boards) for C#
\Samples\C#\DigIO\Properties		Resource files
	AssemblyInfo.cs	File version resource
	Resources.Designer.cs	Application-specific resource manager
	Resources.resx	Configuration of the application-specific resources
	LEDgrey.ico	Image file of the gray status LED
	LEDgreen.ico	Image file of the green status LED
	LEDred.ico	Image file of the red status LED

\Samples\C#\NetManage		C# Sample application for the object dictionary access to external devices via the default SDO for Visual Studio 2008 and .NET 2.0 Framework
	cop.cs	Copy of the CANopen Master API C# main header
	NetManage.csproj	Visual C# 2008 project file
	NetManage.exe	Release build of the sample application
	XatCOP60.dll	CANopen Master API 6
	XatCOP_VCI3.dll	Generic VCI3 firmware
	NetManage.ico	Icon of the application
	NetManage.sln	Visual Studio 2008 project file
	NetManage.csproj.user	Visual C# 2008 project settings
	MainForm.cs	Implementation of the main window of the application
	MainForm.resx	Configuration of the control elements in the main window of the application
	XatBrds.cs	Copy of the IXXAT board type file (CAN boards) for C#

\Samples\C#\NetManage\Properties		Resource files
	AssemblyInfo.cs	File version resource
	FileOpen.bmp	Bitmap for button
	FileSave.bmp	Bitmap for button
	Resources.Designer.cs	Application-specific resource manager
	Resources.resx	Configuration of the application-specific resources

\Samples\Delphi		Delphi sample applications
\Samples\Delphi\DigIO		Delphi sample application for control of an external DS-401 device via its two default-PDOs of length 1
	DigIOdmo.dof	Configuration file of the application
	DigIOdmo.dpr	Delphi project file
	DigIOdmo.exe	Debug build of the sample application
	XatCOP60.dll	CANopen Master API 6
	XatCOP_VCI3.dll	Generic VCI3 firmware
	DigIOdemo.ico	Icon of the application
	DigIOdmo.res	Binary resource file, contains the application icon
	Main.dfm	Configuration of the control elements in the main window of the application
	Main.pas	Implementation of the main window of the application

\Samples\Delphi\NetManage		Delphi sample application for the object dictionary access to external devices via the default SDO
	FileOpen.bmp	Bitmap for button
	FileSave.bmp	Bitmap for button
	Main.dfm	Configuration of the control elements in the main window of the application
	Main.pas	Implementation of the main window of the application
	NetManage.dof	Configuration file of the application
	NetManage.dpr	Delphi project file
	NetManage.exe	Debug build of the sample application
	XatCOP60.dll	CANopen Master API 6
	XatCOP_VCI3.dll	Generic VCI3 firmware
	NetManage.res	Binary resource file, contains the application icon
	NetManage.ico	Icon of the application

\Samples\Delphi.net		Headers for Delphi.net with constants, types and interface descriptions
	cop.pas	Delphi (Pascal) main header of the CANopen Master API
	Copcmd.pas	Delphi (Pascal) header with the command-Opcodes of the CANopen Master API

\Samples\VB.NET		Visual Basic.NET sample applications
\Samples\VB.NET\DigIO		Visual Basic.NET sample application to control an external DS-401 device via its two default PDOs of length 1 for Visual Studio 2008 and .NET 2.0 Framework
	cop.vb	Copy of the CANopen Master API Visual Basic.NET main header
	DigIOdemo.vbproj	Visual Basic 2008 project file
	DigIOdemo.exe	Release build of the sample application
	XatCOP60.dll	CANopen Master API 6
	XatCOP_VCI3.dll	Generic VCI3 firmware
	DigIOdemo.ico	Icon of the application
	DigIOdemo.sln	Visual Studio 2008 project file
	DigIOdemo.vbproj.user	Visual Basic 2008 project settings
	MainForm.vb	Implementation of the main window of the application
	MainForm.resx	Configuration of graphical control elements in the main window of the application
	MainForm.Designer.vb	Configuration of the control elements in the main window of the application
	LED.vb	Implementation of a round colored status LED as a user-defined control element
	LED.resx	Configuration of the control elements for user-defined colored status LED
	XatBrds.vb	Copy of the IXXAT board type file (CAN boards) for Visual Basic .NET
\Samples\C#\DigIO\Properties		Resource files
	app.manifest	User Accound Control definitions
	AssemblyInfo.vb	File version resource
	LEDgrey.ico	Image file of the gray status LED
	LEDgreen.ico	Image file of the green status LED
	LEDred.ico	Image file of the red status LED

\Samples\LabView		LabView examples
	IXXAT CANio500.lvproj	LabView example project for controlling of IXXAT CANopen I/O-module CANio500
	Readme.txt	Note for using the LabView CLN library

\Samples\LabView\CLN_IXXAT_Master_API_Functions		Library of virtual instruments (.vi) using Call-Library-Function nodes für the single Master API functions
	COP_Custom_Error_Codes.vi	Translates the MasterAPI error codes to a LabView compliant range using offset 6100
	COP_AddNode.vi	
	COP_ChangeNodeParameter.vi	
	COP_CheckSync.vi	
	COP_CreatePDO.vi	
	COP_CreateSDO.vi	
	COP_DefSyncObj.vi	
	COP_DeleteNode.vi	
	COP_DeletePDO.vi	
	COP_DisableSync.vi	
	COP_EnableSync.vi	
	COP_EnterPreOperational.vi	
	COP_GetBoardInfo.vi	
	COP_GetEmergencyObj.vi	
	COP_GetEmergencyObj_S.vi	
	COP_GetEvent.vi	
	COP_GetNodeInfo.vi	
	COP_GetNodeState.vi	
	COP_GetPDOInfo.vi	
	COP_GetSDOInfo.vi	
	COP_GetStatus.vi	
	COP_GetSyncInfo.vi	
	COP_GetTimeStampObj.vi	
	COP_InitBoard.vi	
	COP_InitInterface.vi	
	COP_InitTimeStampObj.vi	
	COP_ReadPDO.vi	
	COP_ReadPDO_S.vi	
	COP_ReadSDO.vi	
	COP_ReleaseBoard.vi	
	COP_RequestPDO.vi	
	COP_Reset_DLL.vi	
	COP_ResetComm.vi	
	COP_ResetNode.vi	
	COP_SearchNode.vi	
	COP_SetCommTimeOut.vi	
	COP_SetEmcyIdentifier.vi	
	COP_SetSDOTimeOut.vi	
	COP_SetSyncDivisor.vi	
	COP_StartNode.vi	
	COP_StartStopTSObj.vi	
	COP_StopNode.vi	
	COP_WritePDO.vi	
	COP_WritePDO_S.vi	

	COP_WriteSDO.vi	
	IXXAT_Master_API.mnu	All these Master API functions as LabView palette
	XatCOP60.dll	CANopen Master API 6
	XatCOP_VCI3.dll	Generic VCI3 firmware
\Samples\LabView\CLN_IXXAT_Master_API_Functions\AdditionalFeatures		Library of virtual instruments (.vi) using Call-Library-Function nodes für the single Master API functions of Flying Master additional feature
	COP_ConfigFlyMaster.vi	
	COP_GetStatusFlyMasterNeg.vi	
	COP_StartFlyMaster.vi	
\Samples\LabView\CLN_IXXAT_Master_API_Functions\LSS		Library of virtual instruments (.vi) using Call-Library-Function nodes für the single Master API functions of LSS services
	COP_LSS_ActivateBitTiming.vi	
	COP_LSS_ConfigBitTiming.vi	
	COP_LSS_ConfigNodeID.vi	
	COP_LSS_Fastscan.vi	
	COP_LSS_IdentifyNonConfRemote Slaves.vi	
	COP_LSS_IdentifyRemoteSlaves.vi	
	COP_LSS_InquireAddress.vi	
	COP_LSS_InquireNodeID.vi	
	COP_SetLSSTimeOut.vi	
\Samples\LabView\CLN_IXXAT_Master_API_Functions\special		Library of virtual instruments (.vi) using Call-Library-Function nodes für the single Master API functions for experts
	COP_PutSDO.vi	
	COP_GetSDO.vi	
	COP_CancelSDO.vi	

\Samples\LabView\custom		Manufacturer specific additions to the LabView environment
	IXXAT_Master_API-errors.txt	Master API error codes with LabView-compliant offset 6100

\Samples\LabView\examples		LabView examples
	example_find_all_nodes.vi	
	IXXAT CANio500.vi	
	pdo_statt_lss.vi	
	read_sdo_predefined.vi	
	write_sdo_predefined.vi	
\Samples\LabView\complex_functions_based_on_API		Sub-vis used by the LabView samples and the Master API CLN library (required)
	config_baudrate.vi	
	config_nodeID.vi	
	convert_CAN_to_sdo_value.vi	
	convert_sdo_value_to_CAN.vi	
	find_all_nodes.vi	

	full_scan_different_busspeed.vi inquire_LSSaddress.vi Translate_error_message.vi Translate_event_type.vi	

\Tools\		Utilities
	XCFlash.exe	VCI2 flash programmer for IXXAT CAN interface boards
	ucii161f.H86	VCI2 flash firmware for use with iPCI-XC16/PCI CAN board
	iPC-I_XC16_PCI_FLASH.H86	VCI3 flash firmware for use with iPCI-XC16/PCI CAN board
	COP_i161xc_2ch_flash.H86	VCI2 CANopen Master API 6 dual channel flash firmware for use with iPCI-XC16 CAN board
	COP_i161xc_2ch_flash_vci3.H86	VCI3 CANopen Master API 6 dual channel flash firmware for use with iPCI-XC16 CAN board

Appendix D - Data structures of the command queues

Via the command queues the CANopen Master Firmware is configured and parameterized on the CAN-board. Almost all functions of the CANopen Master API are prepared within the CANopen Master API DLL `XatCOP60.dll` to operations of type `COP_t_Message`, placed in the transmit-command queue and await processing or acknowledgement by the firmware. This is also done as `COP_t_Message`. The definitions of the supported command records are given in the `copcmd` header.

The record COP_t_Message

`COP_t_Message` is the central structure for communication between the Windows side and the CANopen Master Firmware. Here both sides enter their data to be transmitte before the block is written in the command queues. The PC-side compiles one Request (`REQ`) in each case and enters it in the transmit-command queue, whereupon the Master Firmware responds with a Confirmation (`CON`) in the receive-command queue. The Master Firmware does not enter any blocks in the receive-command queue itself but reacts only to blocks from the transmit-command queue.

`COP_t_Message` is composed of a header, containing the Opcode and the size of the occupied memory, as well as an operation-dependent data structure:

`COP_t_Message` Alignment: 1 byte

Field	Type	Meaning
`hd`	`COP_t_Header`	Header of the operation, in which the Opcode and the length of the parameter block are specified.
`pm`	`COP_t_Par`	Operation-dependent parameter block

`COP_t_Header` contains the variables for the operation- or confirmation opcode and the length of the parameter block, which follows immediately in `COP_t_Message` as `COP_t_Par`:

`COP_t_Header` Alignment: 1 byte

Field	Type	Meaning
`cmd`	`WORD`	Operation- or Confirmation opcode
`size`	`WORD`	Size of the operation-dependent parameter block
`SeqNo`	`WORD`	Sequence number allocated by requester for validation of the transmission path
`Walign`	`WORD`	Alignment filler

COP_t_Par is not a fixed record but a union of structures, as the structures vary from operation to operation. In order to facilitate access, any of the following types are entered in the parameter block of a message:

union COP_t_Par Alignment: 1 byte

Union element	Type
t_TestCmd_Con	COP_t_TESTCMD_CON
t_Status_Con	COP_t_STATUS_CON
t_Init_Interface_Req	COP_t_INIT_INTERFACE_REQ
t_Init_Interface_Con	COP_t_INIT_INTERFACE_CON
t_FW_Info_Con	COP_t_FW_INFO_CON
t_Set_UserBittiming_Req	COP_t_SET_USERBITTIMING_REQ
t_Set_UserBittiming_Con	COP_t_SET_USERBITTIMING_CON
t_Add_Node_Req	COP_t_ADD_NODE_REQ
t_Add_Node_Con	COP_t_ADD_NODE_CON
t_Delete_Node_Req	COP_t_DELETE_NODE_REQ
t_Delete_Node_Con	COP_t_DELETE_NODE_CON
t_Search_Node_Req	COP_t_SEARCH_NODE_REQ
t_Search_Node_Con	COP_t_SEARCH_NODE_CON
t_Set_Operational_Req	COP_t_SET_OPERATIONAL_REQ
t_Set_Operational_Con	COP_t_SET_OPERATIONAL_CON
t_Set_Preopertnl_Req	COP_t_SET_PREOPERTNL_REQ
t_Set_Preopertnl_Con	COP_t_SET_PREOPERTNL_CON
t_Reset_Comm_Req	COP_t_RESET_COMM_REQ
t_Reset_Comm_Con	COP_t_RESET_COMM_CON
t_Reset_Node_Req	COP_t_RESET_NODE_REQ
t_Reset_Node_Con	COP_t_RESET_NODE_CON
t_Set_Prepared_Req	COP_t_SET_PREPARED_REQ
t_Set_Prepared_Con	COP_t_SET_PREPARED_CON
t_Get_Node_State_Req	COP_t_GET_NODE_STATE_REQ
t_Get_Node_State_Con	COP_t_GET_NODE_STATE_CON
t_Ident_Node_Info_Req	COP_t_GET_NODE_INFO_REQ
t_Ident_Node_Info_Con	COP_t_GET_NODE_INFO_CON
t_Change_Node_Param_Req	COP_t_CHANGE_NODE_PARAM_REQ
t_Change_Node_Param_Con	COP_t_CHANGE_NODE_PARAM_CON
t_Create_PDO_Req	COP_t_CREATE_PDO_REQ
t_Create_PDO_Con	COP_t_CREATE_PDO_CON
t_Get_PDO_Info_Req	COP_t_GET_PDO_INFO_REQ
t_Get_PDO_Info_Con	COP_t_GET_PDO_INFO_CON
t_Create_SDO_Req	COP_t_CREATE_SDO_REQ
t_Create_SDO_Con	COP_t_CREATE_SDO_CON
t_Def_SyncObj_Req	COP_t_DEF_SYNCOBJ_REQ
t_Def_SyncObj_Con	COP_t_DEF_SYNCOBJ_CON
t_Get_Sync_Info_Con	COP_t_GET_SYNC_INFO_CON
t_Set_SyncDivisor_Req	COP_t_SET_SYNCDIVISOR_REQ
t_Set_SyncDivisor_Con	COP_t_SET_SYNCDIVISOR_CON
T_Enable_Sync_Req	COP_t_ENABLE_SYNC_REQ
t_Enable_Sync_Con	COP_t_ENABLE_SYNC_CON
t_Disable_Sync_Req	COP_t_DISABLE_SYNC_REQ
t_Disable_Sync_Con	COP_t_DISABLE_SYNC_CON
t_En_Dis_TSObj_Req	COP_t_EN_DIS_TSOBJ_REQ
t_En_Dis_TSObj_Con	COP_t_EN_DIS_TSOBJ_CON
t_Set_SDO_TmOut_Req	COP_t_SET_SDO_TMOUT_REQ

t_Set_SDO_TmOut_Con	COP_t_SET_SDO_TMOUT_CON
t_Get_TS_Obj_Con	COP_t_GET_TS_OBJ_CON
t_Get_SDO_Info_Req	COP_t_GET_SDO_INFO_REQ
t_Get_SDO_Info_Con	COP_t_GET_SDO_INFO_CON
t_Set_Emcy_ID_Req	COP_t_SET_EMCY_ID_REQ
t_Set_Emcy_ID_Con	COP_t_SET_EMCY_ID_CON
t_Request_PDO_Req	COP_t_REQUEST_PDO_REQ
t_Request_PDO_Con	COP_t_REQUEST_PDO_CON
t_Cancel_SDO_Req	COP_t_CANCEL_SDO_REQ
t_Cancel_SDO_Con	COP_t_CANCEL_SDO_CON
t_Start_Master_Neg_Con	COP_t_START_MASTER_NEG_CON
t_Config_Fly_Master_Req	COP_t_CONFIG_FLY_MASTER_REQ
t_Config_Fly_Master_Con	COP_t_CONFIG_FLY_MASTER_CON
t_Get_Status_Master_Neg_Con	COP_t_GET_STATUS_MASTER_NEG_CON
t_Req_Lmt_Config_Node_Id_Macro	COP_t_REQ_LMT_CONFIG_NODE_ID_MACRO
t_Con_Lmt_Config_Node_Id_Macro	COP_t_CON_LMT_CONFIG_NODE_ID_MACRO
t_Req_Lmt_Config_Bit_Timing_Macro	COP_t_REQ_LMT_CONFIG_BIT_TIMING_MACRO
t_Con_Lmt_Config_Bit_Timing_Macro	COP_t_CON_LMT_CONFIG_BIT_TIMING_MACRO
t_Req_Lmt_Inquire_Address_Macro	COP_t_REQ_LMT_INQUIRE_ADDRESS_MACRO
t_Con_Lmt_Inquire_Address_Macro	COP_t_CON_LMT_INQUIRE_ADDRESS_MACRO
t_Req_Lmt_Identify_Slave_Macro	COP_t_REQ_LMT_IDENTIFY_SLAVE_MACRO
t_Con_Lmt_Identify_Slave_Macro	COP_t_CON_LMT_IDENTIFY_SLAVE_MACRO
t_Req_Lss_Config_Node_Id_Macro	COP_t_REQ_LSS_CONFIG_NODE_ID_MACRO
t_Con_Lss_Config_Node_Id_Macro	COP_t_CON_LSS_CONFIG_NODE_ID_MACRO
t_Req_Lss_Config_Bit_Timing_Macro	COP_t_REQ_LSS_CONFIG_BIT_TIMING_MACRO
t_Con_Lss_Config_Bit_Timing_Macro	COP_t_CON_LSS_CONFIG_BIT_TIMING_MACRO
t_Req_Lss_Activate_Bit_Timing_Macro	COP_t_REQ_LSS_ACTIVATE_BIT_TIMING_MACRO
t_Con_Lss_Activate_Bit_Timing_Macro	COP_t_CON_LSS_ACTIVATE_BIT_TIMING_MACRO
t_Req_Lss_Identify_Slave_Macro	COP_t_REQ_LSS_IDENTIFY_SLAVE_MACRO
t_Con_Lss_Identify_Slave_Macro	COP_t_CON_LSS_IDENTIFY_SLAVE_MACRO
t_Req_Lss_Inquire_Address_Macro	COP_t_REQ_LSS_INQUIRE_ADDRESS_MACRO
t_Con_Lss_Inquire_Address_Macro	COP_t_CON_LSS_INQUIRE_ADDRESS_MACRO
t_Req_Lss_Inquire_Node_Id_Macro	COP_t_REQ_LSS_INQUIRE_NODE_ID_MACRO
t_Con_Lss_Inquire_Node_Id_Macro	COP_t_CON_LSS_INQUIRE_NODE_ID_MACRO
t_Req_Lss_Identify_Non_Config_Slave_Macro	COP_t_REQ_LSS_IDENTIFY_NON_CONFIG_SLAVE_MACRO
t_Con_Lss_Identify_Non_Config_Slave_Macro	COP_t_CON_LSS_IDENTIFY_NON_CONFIG_SLAVE_MACRO

t_Req_LSS_Set_TimeOut	COP_t_REQ_LSS_SET_TIMEOUT
t_Con_LSS_Set_TimeOut	COP_t_CON_LSS_SET_TIMEOUT
t_Req_LSS_Fastscan	COP_t_REQ_LSS_FASTSCAN
t_Con_LSS_Fastscan	COP_t_CON_LSS_FASTSCAN
para[COP_k_SIZE_PARA_BUF]	BYTE[]

Command Opcodes

The operation- and confirmation opcodes specify how the parameter block is to be interpreted. To facilitate machine processing and orientation in the API, the code numbers are divided into bit-groups (Figure 0-1). The module identifier describes the Opcode category and the Service Opcode identifies the service itself.

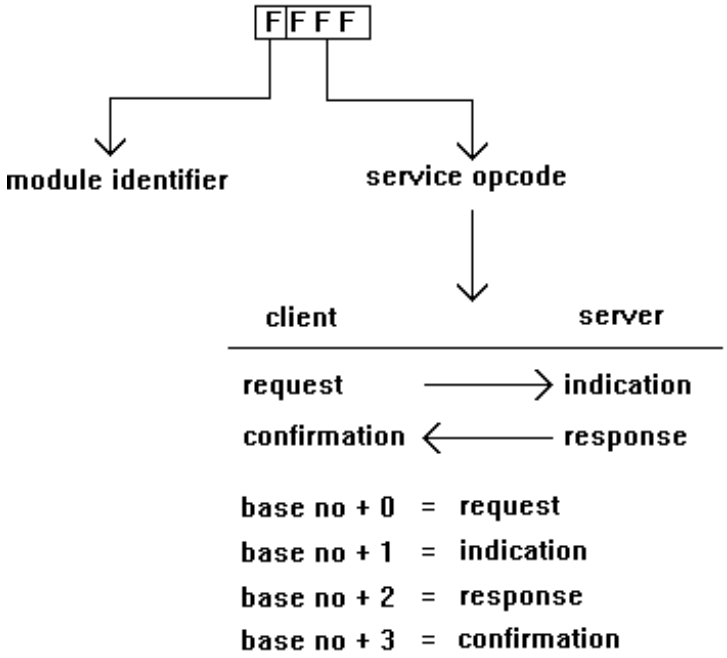

Fig. D-1: Subdivision of the message block Opcodes

Basic API-functions:

Function name	Opcode name	Opcode value
COP_InitInterface	COP_k_INIT_INTERFACE_REQ	0x0008
	COP_k_INIT_INTERFACE_CON	0x000b
COP_GetBoardInfo	COP_k_FW_INFO_REQ	0x000c
	COP_k_FW_INFO_CON	0x000f
COP_GetStatus	COP_k_STATUS_REQ	0x0004
	COP_k_STATUS_CON	0x0007
COP_TestCommand	COP_k_TESTCMD_REQ	0x0000
	COP_k_TESTCMD_CON	0x0003

-	COP_k_SHUTDOWN_REQ	0x0010
	COP_k_SHUTDOWN_CON	0x0013
COP_SetUserBittiming	COP_k_SET_USERBITTIMING_REQ	0x0014
	COP_k_SET_USERBITTIMING_CON	0x0017

Functions for the network management:

Function name	Opcode name	Opcode value
COP_AddNode	COP_k_ADD_NODE_REQ	0x1000
	COP_k_ADD_NODE_CON	0x1003
COP_DeleteNode	COP_k_DELETE_NODE_REQ	0x1008
	COP_k_DELETE_NODE_CON	0x100b
COP_SearchNode	COP_k_SEARCH_NODE_REQ	0x1004
	COP_k_SEARCH_NODE_CON	0x1007
COP_GetNodeInfo	COP_k_GET_NODE_INFO_REQ	0x1034
	COP_k_GET_NODE_INFO_CON	0x1037
COP_ChangeNodeParameter	COP_k_CHANGE_NODE_PARAM_REQ	0x102c
	COP_k_CHANGE_NODE_PARAM_CON	0x102f
COP_SetEmcyIdentifier	COP_k_SET_EMCY_ID_REQ	0x3060c
	COP_k_SET_EMCY_ID_CON	0x3063f
COP_StartNode	COP_k_SET_OPERATIONAL_REQ	0x100c
	COP_k_SET_OPERATIONAL_CON	0x100f
COP_StopNode	COP_k_SET_PREPARED_REQ	0x1018
	COP_k_SET_PREPARED_CON	0x101b
COP_ResetComm	COP_k_RESET_COMM_REQ	0x101c
	COP_k_RESET_COMM_CON	0x101f
COP_ResetNode	COP_k_RESET_NODE_REQ	0x1020
	COP_k_RESET_NODE_CON	0x1023
COP_EnterPreOperational	COP_k_SET_PREOPERTNL_REQ	0x1010
	COP_k_SET_PREOPERTNL_CON	0x1013
COP_GetNodeState	COP_k_GET_NODE_STATE_REQ	0x1024
	COP_k_GET_NODE_STATE_CON	0x1027
COP_ConfigFlyMaster	COP_k_CONFIG_FLY_MASTER_REQ	0x1050
	COP_k_CONFIG_FLY_MASTER_CON	0x1053
COP_StartFlyMaster	COP_k_START_MASTER_NEG_REQ	0x1054
	COP_k_START_MASTER_NEG_CON	0x1057
COP_GetStatusFlyMasterNeg	COP_k_GET_STATUS_MASTER_NEG_REQ	0x1058
	COP_k_GET_STATUS_MASTER_NEG_CON	0x105b
-	COP_k_CONFIG_SDM_REQ	0x105c
	COP_k_CONFIG_SDM_CON	0x105f
-	COP_k_START_SDM_REQ	0x1060

	COP_k_START_SDM_CON	0x1063

CANopen-object management:

Function name	Opcode name	Opcode value
COP_CreatePDO	COP_k_CREATE_PDO_REQ	0x3000
	COP_k_CREATE_PDO_CON	0x3003
COP_GetPDOInfo	COP_k_GET_PDO_INFO_REQ	0x3004
	COP_k_GET_PDO_INFO_CON	0x3007
COP_CreateSDO	COP_k_CREATE_SDO_REQ	0x3044
	COP_k_CREATE_SDO_CON	0x3047
COP_GetSDOInfo	COP_k_GET_SDO_INFO_REQ	0x3054
	COP_k_GET_SDO_INFO_REQ	0x3057
COP_SetSDOTimeOut	COP_k_SET_SDO_TMOUT_REQ	0x3040
	COP_k_SET_SDO_TMOUT_CON	0x3043
COP_DefSyncObj	COP_k_DEF_SYNCHOBJ_REQ	0x3008
	COP_k_DEF_SYNCHOBJ_CON	0x300b
COP_SetSyncDivisor	COP_k_SET_SYNCDIVISOR_REQ	0x3048
	COP_k_SET_SYNCDIVISOR_CON	0x304b
COP_GetSyncInfo	COP_k_GET_SYNC_INFO_REQ	0x300c
	COP_k_GET_SYNC_INFO_CON	0x300f
COP_EnableSync	COP_k_ENABLE_SYNCH_REQ	0x3014
	COP_k_ENABLE_SYNCH_CON	0x3017
COP_DisableSync	COP_k_DISABLE_SYNCH_REQ	0x3018
	COP_k_DISABLE_SYNCH_CON	0x301b
COP_StartStopTSObj	COP_k_EN_DIS_TS_OBJ_REQ	0x303c
	COP_k_EN_DIS_TS_OBJ_CON	0x303f
COP_GetTimeStampObj	COP_k_GET_TS_OBJ_REQ	0x3050
	COP_k_GET_TS_OBJ_CON	0x3053
COP_CreateSpdTmObj	COP_k_CREATE_SPDTMOBJ_REQ	0x302c
	COP_k_CREATE_SPDTMOBJ_CON	0x302f
COP_SetSpeedTime	COP_k_SET_SPEEDTIME_REQ	0x3030
	COP_k_SET_SPEEDTIME_CON	0x3033
COP_StartStopSpdTmObj	COP_k_EN_DIS_SPDTMOBJ_REQ	0x3034
	COP_k_EN_DIS_SPDTMOBJ_CON	0x3037

CANopen communication:

Function name	Opcode name	Opcode value
COP_ReadPDO COP_ReadPDO_S	COP_k_RX_PDO_IND	0x2005
COP_RequestPDO	COP_k_REQUEST_PDO_REQ	0x2014
	COP_k_REQUEST_PDO_CON	0x2017
COP_WritePDO COP_WritePDO_S	COP_k_WRITE_PDO_REQ	0x0000
COP_ReadSDO	COP_k_READ_SDO_REQ	0x2000
	COP_k_READ_SDO_CON	0x2003
	COP_k_BLOCKREAD_SDO_REQ	0x2020
	COP_k_BLOCKREAD_SDO_CON	0x2023
COP_WriteSDO	COP_k_WRITE_SDO_REQ	0x2004
	COP_k_WRITE_SDO_CON	0x2007
	COP_k_BLOCKWRITE_SDO_REQ	0x2024
	COP_k_BLOCKWRITE_SDO_CON	0x2027
COP_CancelSDO	COP_k_CANCEL_SDO_REQ	0x2028
	COP_k_CANCEL_SDO_CON	0x202b
COP_GetEmergencyObj COP_GetEmergencyObj_S	COP_k_EMERGENCY_OBJ_IND	0x2011
COP_GetEvent	COP_k_EVENT_IND	0x1031

Of this category, only COP_CancelSDO works on the command queue. All other functions use the Opcodes given in the table but they work on the corresponding data queues.

LMT services:

Function name	Opcode name	Opcode value
COP_LMT_ConfigNode COP_LMT_ConfigModuleID	COP_k_REQ_LMT_CONFIG_NODE_ID_MACRO	0x4004
	COP_k_CON_LMT_CONFIG_NODE_ID_MACRO	0x4007
COP_LMT_GetAddress	COP_k_REQ_LMT_INQUIRE_ADDRESS_MACRO	0x4000
	COP_k_CON_LMT_INQUIRE_ADDRESS_MACRO	0x4003
COP_LMT_ConfigNode	COP_k_REQ_LMT_CONFIG_BIT_TIMING_MACRO	0x4008
	COP_k_CON_LMT_CONFIG_BIT_TIMING_MACRO	0x400b
COP_LMT_IdentifyRemoteSlaves	COP_k_REQ_LMT_IDENTIFY_SLAVE_MACRO	0x400c
	COP_k_CON_LMT_IDENTIFY_SLAVE_MACRO	0x400f

LSS services:

Function name	Opcode name	Opcode value

COP_LSS_ConfigNodeID	COP_k_REQ_LSS_CONFIG_NODE_ID_MACRO	0x4020
	COP_k_CON_LSS_CONFIG_NODE_ID_MACRO	0x4023
COP_LSS_ConfigBitTiming	COP_k_REQ_LSS_CONFIG_BIT_TIMING_MACRO	0x4024
	COP_k_CON_LSS_CONFIG_BIT_TIMING_MACRO	0x4027
COP_LSS_ActivateBitTiming	COP_k_REQ_LSS_ACTIVATE_BIT_TIMING_MACRO	0x4028
	COP_k_CON_LSS_ACTIVATE_BIT_TIMING_MACRO	0x402b
COP_LSS_IdentifyRemoteSlaves	COP_k_REQ_LSS_IDENTIFY_SLAVE_MACRO	0x402c
	COP_k_CON_LSS_IDENTIFY_SLAVE_MACRO	0x402f
COP_LSS_InquireAddress	COP_k_REQ_LSS_INQUIRE_ADDRESS_MACRO	0x4030
	COP_k_CON_LSS_INQUIRE_ADDRESS_MACRO	0x4033
COP_LSS_InquireNodeID	COP_k_REQ_LSS_INQUIRE_NODE_ID_MACRO	0x4034
	COP_k_CON_LSS_INQUIRE_NODE_ID_MACRO	0x4037
COP_LSS_IdentifyNonConfRemoteSlaves	COP_k_REQ_LSS_IDENTIFY_NON_CONFIG_SLAVE_MACRO	0x4038
	COP_k_CON_LSS_IDENTIFY_NON_CONFIG_SLAVE_MACRO	0x403b
COP_LSS_Fastscan	COP_k_REQ_LSS_FASTSCAN	0x4040
	COP_k_CON_LSS_FASTSCAN	0x4043
COP_SetLSSTimeOut	COP_k_REQ_LSS_SET_TIMEOUT	0x403c
	COP_k_CON_LSS_SET_TIMEOUT	0x403f

Appendix E - Differences to version 5.x

New functions

These eight functions have been added to the CANopen Master API 6:

Removed functions

Deleted functions

This function is no longer included in the CANopen Master API 6:

COP_IdentifyRemNode()

Inapplicable functions

None

Altered functions

Renamed functions

None

Functions with altered parameter set

With these functions of the CANopen Master API 6, the parameter set was altered. By way of comparison, the new and old syntax are shown opposite each other.

Old syntax	New syntax
COP_CreatePDO (COP_t_HANDLE boardhdl, BYTE node_no, BYTE pdo_no, BYTE type, BYTE mode, BYTE length, WORD cob_id);	**COP_CreatePDO** (COP_t_HANDLE boardhdl, BYTE node_no, BYTE pdo_no, BYTE type, *BYTE mode,* BYTE length, WORD CANid);
COP_DefSyncObj (COP_t_HANDLE boardhdl, WORD sync_period, WORD sync_window);	**COP_DefSyncObj** (COP_t_HANDLE boardhdl, WORD sync_period, WORD sync_window, *BYTE CounterOverflow*);
COP_EnableSync (COP_t_HANDLE boardhdl, *WORD cycle_count*, BYTE mode);	**COP_EnableSync** (COP_t_HANDLE boardhdl, *BYTE mode*);
COP_CheckSync (COP_t_HANDLE boardhdl);	**COP_CheckSync** (COP_t_HANDLE boardhdl, *BYTE* SyncCounter*);
COP_ReadPDO (COP_t_HANDLE boardhdl, BYTE* node_no, BYTE* pdo_no, BYTE* rxlen, BYTE* rxdata);	**COP_ReadPDO** (COP_t_HANDLE boardhdl, BYTE* node_no, BYTE* pdo_no, BYTE* rxlen, BYTE* rxdata *BYTE* SyncCounter*);
COP_ReadPDO_S (COP_t_HANDLE boardhdl, COP_t_RX_PDO* sp_pdo); typedef struct { BYTE node_no; BYTE pdo_no; BYTE length; BYTE reserved; BYTE a_data[8]; } COP_t_RX_PDO;	**COP_ReadPDO_S** (COP_t_HANDLE boardhdl, COP_t_RX_PDO* sp_pdo); typedef struct { BYTE node_no; BYTE pdo_no; BYTE length; *BYTE SyncCounter;* BYTE a_data[8]; } COP_t_RX_PDO;

Appendix F - CANopen-specific aspects

Processing of synchronous PDOs

The treatment and processing of synchronous PDOs by the CANopen-Master Firmware is described.

TPDOs

PDO-data for synchronous PDOs transfered with `COP_WritePDO()` are stored in an internal buffer for the individual PDO. With each Sync-object all synchronous TPDOs are checked to see whether data are present in the buffer. For the application this means that with cyclic synchronous PDOs the data do not have to be transferred separately with each Sync-event.

CANopen Master API 6 supports all PDO transmission types. The transmission type is configured, according to the coding of subindex2 of the CANopen PDO communication parameters in the object dictionary, in argument `mode` of function `COP_CreatePDO()`.

RPDOs

Received PDOs, which are defined as synchronous, are buffered in a separate internal RPDO-Queue of the Master Firmware until the next Sync-object. Not until after transmitting the Sync object, all PDOs received in the meantime and stored in this internal RPDO-Queue are transfered via the 'correct' RPDO-Queue and can be read with `COP_ReadPDO()`.

Node guarding and node states

If a device signals an unexpected node state in the course of node guarding, the Master Firmware generates a network event of type COP_k_NMT_EVT with the event cause COP_k_NMT_GUARDERR or COP_k_NMT_HEARTBEATERR in the second return parameter evt_data1 and the node-ID in the third return parameter evt_data2 of the function COP_GetEvent(). An unexpected node state is given as the node state signaled by the slave is different to the one set by the client application with the NMT functions, and also as any slave is not in Pre-Operational state after its Bootup. The same network events are generated when the node involved is lost, i.e. no longer reacts at all to the Guarding telegram or no longer transmits Heartbeat messages. For a better distinction of the various error scenarios, the fourth return parameter evt_data3 contains the unexpected node state.

Since the firmware keeps track of all the current node states, the application does not have to process all the network events. To get the NMT state of a registered node, call function COP_GetNodeState().

The node guarding starts automatically on registration of the node, i.e. COP_AddNode().

For reception of bootup indications by a network event of type COP_k_NMT_EVT with the event cause COP_k_NMT_BOOTIND in evt_data1 and the node-ID in evt_data2 as well as for reception of emergency messages (COP_GetEmergencyObj) *no* node registration is required. Both these two message types are received invariably regardless of the node-ID.

The node guarding is switched off indirectly by calling the function COP_ChangeNodeParameter(), i.e. overwriting the former monitoring interval by 0.

Appendix G - Frequent sources of errors

To facilitate debugging, there follow some tips for troubleshooting.

Presetting and initialising the CAN board

For unique identification of the desired CAN board there are two arguments with COP_InitBoard() which are strictly checked by Master API DLL and thus must be initialised properly before the call:

```
GUID  boardtype = GUID_CANATNET2_DEVICE;
GUID  boardID   = COP_1stBOARD;
WORD  wBoardhdl;

short res       = COP_InitBoard( &wBoardhdl, &boardtype, &boardID, 0 );
```

If, for example, the explicit initialisation of **boardID** is missing, the compiler might do its own initialisation (typically by setting the variable's memory to zero), or the memory might even contain random data. In both cases for Master API the value is not a valid board identification, which is why the function call will result in **BER_k_BOARD_NOT_FOUND**

The unique board identification of the local CAN board is delivered by the function and normally is equate to the serial number:

```
if( BER_k_OK == res )
{
    char    sz[32] = {0};

    sprintf_s( sz, sizeof(sz), "%s", &boardID );    //  e.g. HW800511
}
```

Reading out receive-data queues

The four receive-data queues RPDO-Queue, EMCY-Queue, Event-Queue and Sync-Queue must be read continually and completely until empty because their capacity is relatively small. The Queues can be read asynchronously or synchronously (Polling). With the asynchronous method the functions COP_DefineCallbacks() or COP_DefineMsgRPDO COP_DefineMsgEvent COP_DefineMsgEmergency COP_DefineMsgSync shall be called for initialization, so that you can be informed by the firmware as soon as a receive object has been entered in the corresponding data queue (see also the reference of these function family in section 5.1).

Especially when Callback functions or Windows messages are defined, the receive objects must be read out of the corresponding queue with the functions COP_ReadPDO(), COP_GetEmergencyObj(), COP_GetEvent() and

COP_CheckSync(). Failing this, after a while each anew arriving CANopen object will cause a queue overrun notification (COP_GetEvent).

In the following you will find pseudo-code, which describes how the individual Queues are to be read out:

RPDO-Queue:

```
do
{
    iRes = COP_ReadPDO( ... );
    if( COP_k_OK == iRes )
        ...
}
while( COP_k_OK == iRes );
```

EMCY-Queue:

```
do
{
    iRes = COP_GetEmergencyObj( ... );
    if( COP_k_OK == iRes )
        ...
}
while( COP_k_OK == iRes );
```

Event-Queue:

```
do
{
    iRes = COP_GetEvent( ... );
    if( COP_k_OK == iRes )
        ...
}
while( iRes == COP_k_OK );
```

Sync-Queue:

```
do
{
    iRes = COP_CheckSync( ... );
    if( COP_k_OK == iRes )
        ...
}
while( iRes == COP_k_OK );
```

Appendix H - Timer resolutions and value ranges

The following table lists the value ranges and resolutions of all those Master API functions that work with parameterizable delay times.

	value range [ms]	resolution [ms]
Sync-object: `COP_DefSyncObj()` Cycle time (`sync_period`)	2 .. 65280	1
Sync-object: `COP_DefSyncObj()` Synchronization window (`sync_window`)	2 .. 65280	1
Heartbeat / Guarding: `COP_AddNode()` / `COP_ChangeNodeParameter()` Monitoring time (`GuardHeartbeatTime`)	5 .. 32767	1
Master Initialisation: `COP_InitInterface()` Heartbeattime (`hbtime`)	5 .. 32767	1
SDO Timeout: `COP_SetSDOTimeOut()`	5 .. 32767	1
Central time information: `COP_StartStopTSObj()` Cycle time (`cycle`)	2 .. 65280	1

Flying Master additional functionality:
`COP_ConfigFlyMaster()`
`wDetectionTimeout`,
`wNegotiationDelay`,
`wPrioritytimeslot`,
`wNodeTimeslot`,
`wCycletimeCd`,
`wCycletimeTimeoutHbeat`
	
	5 .. 32767
	5 .. 32767
	5 .. 32767
	5 .. 32767
	5 .. 32767
	5 .. 32767
		
		1
		1
		1
		1
		1
		1

LSS Timeout: `COP_SetLSSTimeOut()`	5 .. 32767	1