Ixxat®
BY HMS NETWORKS

CANopen Master API

Software Version 6.3

SOFTWARE DESIGN GUIDE

4.12.0132.20000 1.0 en-US ENGLISH

HMS Connecting Devices™

Important User Information

Disclaimer

The information in this document is for informational purposes only. Please inform HMS Industrial Networks of any inaccuracies or omissions found in this document. HMS Industrial Networks disclaims any responsibility or liability for any errors that may appear in this document.

HMS Industrial Networks reserves the right to modify its products in line with its policy of continuous product development. The information in this document shall therefore not be construed as a commitment on the part of HMS Industrial Networks and is subject to change without notice. HMS Industrial Networks makes no commitment to update or keep current the information in this document.

The data, examples and illustrations found in this document are included for illustrative purposes and are only intended to help improve understanding of the functionality and handling of the product. In view of the wide range of possible applications of the product, and because of the many variables and requirements associated with any particular implementation, HMS Industrial Networks cannot assume responsibility or liability for actual use based on the data, examples or illustrations included in this document nor for any damages incurred during installation of the product. Those responsible for the use of the product must acquire sufficient knowledge in order to ensure that the product is used correctly in their specific application and that the application meets all performance and safety requirements including any applicable laws, regulations, codes and standards. Further, HMS Industrial Networks will under no circumstances assume liability or responsibility for any problems that may arise as a result from the use of undocumented features or functional side effects found outside the documented scope of the product. The effects caused by any direct or indirect use of such aspects of the product are undefined and may include e.g. compatibility issues and stability issues.

Table of Contents

Page

1 User Guide

Please read the manual carefully. Make sure you fully understand the manual before using the product.

1.1 Target Audience

This manual is intended for software and application developers who are familiar with the various mechanisms and terms of CANopen. For more information see the relevant CANopen specifications that are available from CiA (www.can-cia.org).

1.2 Related Documents

Document	Author
CiA 301 CANopen Application Layer and Communication Profile	CAN in Automation
CiA 302 Additional Application Layer Functions	CAN in Automation
CiA 305 Layer Settings Services and Protocol (LSS)	CAN in Automation
CiA 306 Electronic Data Sheet for CANopen (EDS)	CAN in Automation
CiA 401 CANopen Device Profile for Generic I/O Modules	CAN in Automation

1.3 Document History

Version	Date	Description
1.0	April 2019	Reworked and edited in new design

1.4 Trademark Information

1.5 Conventions

Instructions and results are structured as follows:

- ► instruction 1
- ► instruction 2
  - → result 1
  - → result 2

Lists are structured as follows:

- item 1
- item 2

Bold typeface indicates interactive parts such as connectors and switches on the hardware, or menus and buttons in a graphical user interface.

```
This font is used to indicate program code and other
kinds of data input/output such as configuration scripts.
```

This is a cross-reference within this document: *Conventions, p. 6*

This is an external link (URL): www.hms-networks.com

This is additional information which may facilitate installation and/or operation.

! This instruction must be followed to avoid a risk of reduced functionality and/or damage to the equipment, or to avoid a network security risk.

1.6 Definitions, Acronyms and Abbreviations

Bootup message	Special heartbeat message, the CANopen device signals the local transition from the NMT state *initialization* to the NMT state *pre-operational* , see NMT.
CAN ID	The CAN ID or message identifier identifies a CAN message and defines the message priority. The highest priority ID 0 is reserved for network management services NMT.
CiA	CAN in Automation e. V., organization of CAN bus device manufactures and users
Client SDO	Initiator of an SDO transmission, the SDO client has access to the object dictionary entries of a SDO server (see SDO, OD)
Communication cycle period	Defines the time interval between sequential SYNC objects.
DLL	Data Link Layer
Emergency message	With a high-priority emergency message a device signals the occurrence of a internal device error or the reset of one or all device internal errors.
Error control services	Cyclic monitoring of a CANopen device. Error control can be implemented via node guarding or heartbeat.
Guard time	The NMT master cyclically transmits a request to the NMT slave to transmit its current node status. The request must be answered within the node lifetime. The node lifetime is defined by lifetime factor multiplied with the guard time. The NMT slave guards the NMT master based on the incoming guarding requests, if the guard time entry in the object dictionary of the NMT slave is unequal 0. The reactions to violations of the node guarding are described in the CANopen specification.
Heartbeat message	Heartbeat messages transmit the device NMT state cyclically and independent of request messages from a NMT master.
Inhibit time	A PDO (process data object) can only be transmitted again after the inhibit time is expired.
LSS (Layer Settings Services)	Service element that enables the setting of basic device communication parameters in the physical and in the application layer such as bitrate and node ID.
NMT (Network Management)	Service element of the application layer in the CANopen reference model, that comprises the configuration, initialization and error processing in the network as well as network wide process synchronization. CANopen specifies the four NMT states *initialization*, *pre-operational*, *operational* and *stopped*. NMT commands trigger the state transition of a CANopen node. NMT is CANopen device orientated and follows a master slave structure.
Node ID	Each device in the CAN network is defined by its unique node ID (1–127). The node id is used by the Predefined Connection Set for the predefined CAN identifier allocation. The node ID 0 is reserved for NMT services.
Error control services	Cyclic monitoring of a CANopen device. Node guarding can be implemented via guard time or heartbeat.
OD (Object Dictionary)	The object dictionary is a data structure via that all objects of a CANopen device can be addressed. The object dictionary is subdivided into a communication profile area, a manufacturer specific area and a device and application profile area. 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 signals and output signals, device parameters, function or network variables are made accessible via the network in standardized form. 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 a CAN identifier that is uniquely allocated to the PDO. The meaning of the transmitted data is defined by the identifier in use 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 two data structures per PDO. The PDO producer manages the required data by TPDO data structures, the data to be received by a PDO consumer are managed by RPDO data structures.
PDO communication parameters	The attributes of a PDO are described in the communication parameters. The attributes include transmission type, inhibit time and the CAN ID.
Predefined connection set	Preset allocation of 11 bit CAN IDs based on the node ID and on a function code. The 127 nodes are differentiated via the four lower bits of the identifier. The Predefined Connection Set defines the CAN IDs for the following communication objects: NMT node control, SYNC object, emergency message, time stamp, TPDO 1–4, RPDO 1–4, server SDO and NMT error control (heartbeat, node guarding). 29 bit identifiers are not supported.
RPDO (Receive PDO)	See PDO.
SDO (Service Data Object)	An SDO is a CAN communication object that initializes and parameterizes CANopen devices or transmits 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 indicating 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 to allow access to the entries in its object dictionary. The specification of a server SDO object requires the definition of a CAN identifier for each transmission direction (acknowledged service) and the specification of the corresponding client or server node if dynamic structure of SDO connections is supported.
SYNC object	The SYNC object is used for synchronized data collection, synchronized command strobing and cyclic transmission of process data. The reception of a SYNC object triggers the updating and transmission of synchronous messages. A device, the SYNC producer cyclically transmits the high-priority SYNC object. The SYNC object is completely described with the specification of the CAN ID, the Communication Cycle Period parameter, and the Synchronous Window Length parameter. If a parameter is initialized with 0, it has no effect.
Synchronous window length	Time frame after a SYNC object within which the synchronous PDOs must be transmitted
Time stamp message	The time stamp message enables the resynchronization the local timers, to ensure the 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 synchronous, asynchronous and remote transmission types for PDOs. The transmission of a synchronous PDO is, depending on a SYNC object, acyclic (once) or cyclic (with each reception or after a number of SYNC objects, specified by transmission rate). The transmission of an asynchronous PDO is triggered by a manufacturer specific event or by an event defined via a device profile. Remote transmission occurs only when 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 be received before the PDO is transmitted again.
TPDO	See PDO.

2 System and Software Overview

2.1 Product Description

The CANopen Master Application Programming Interface (API) is a programming library to connect a PC application to a CANopen network.

The CANopen Master API is implemented as a pair of libraries:

- Microsoft Windows: XatCOP60.dll and XatCOP_VCI3.dll
- Linux: libXatCOP60.so and libXatCOP_sock.so, or libXatCOP_ECI.so

Encapsulated in the library is the CANopen Master Firmware. The CANopen Master Firmware independently configures and uses the CAN controller.

Six data queues are used to communicate with the network and two command queues (CMD queues) are used to communicate with the CANopen Master Firmware. The CANopen Master API loads the CANopen Master Firmware into the volatile memory on the interface, sets up the queues and edits the data for the queues.

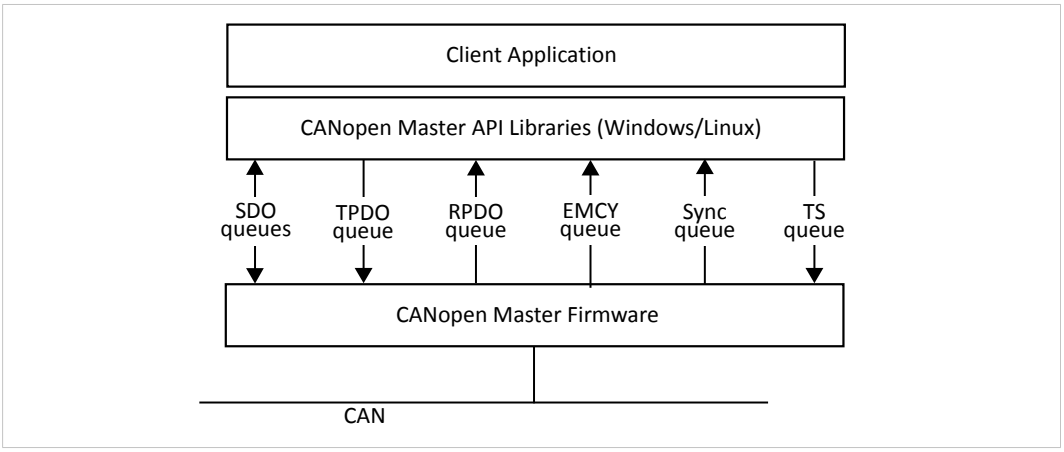

Fig. 1 Communication between CANopen Master API DLL and CANopen Master Firmware

For the direct information exchange with individual network subscribers process data objects (PDOs) and service data objects (SDOs) are available. System services like the synchronization object (Sync) and the central time object (Time Stamp) are provided and device error messages (EMCY, Emergency messages) can be evaluated.

The following features of the CANopen specification CiA 301 V4 are supported:

- NMT node control
- NMT error control
- PDO
- SYNC (including SYNC counter)
- SDO (including block transfer)
- EMCY
- LSS
- Flying NMT master

For these functions the CAN identifiers are allocated according to the Predefined Connection Set. For PDOs and SDOs it is also possible to define alternative CAN identifiers.

Current information about the software that is not included in the manual is available as README files on the data carriers that are included in the scope of delivery.

2.2 Supported CAN Interfaces

To access the CAN bus an Ixxat CAN interface must be installed.

2.2.1 Windows

Interfaces Based on VCI

The included windows library *XatCOP60_VCI3.dll* allows to use all Ixxat CAN interfaces that are based on the VCI, like for example the USB-to-CAN^{V2} and passive Ixxat CAN interfaces.

Active CAN Interfaces

The CANopen Master API supports active CAN interfaces that have a microcontroller to execute the CANopen Master Firmware of the API. For further details about the capacity limits see *Performance Characteristics, p. 106*.

Supported Ixxat CAN Interfaces

PC interface	CAN interface
PCI	CAN-IB400/PCI
PCIexpress	CAN-IB200/PCIe
PCIe 104	CAN-IB230/PCIe 104
Ethernet	CAN@net II/VCI

For information about legacy CAN interfaces contact Ixxat support on www.ixxat.com/support.

2.2.2 Linux

The software requires either the Ixxat SocketCAN driver package or the Ixxat ECI package for CAN interface board access. The drivers do not support ISA CAN boards. For more information about the capacity limits see *Performance Characteristics, p. 106*. Ixxat CAN FD interface boards are supported in CAN compatibility mode. The CANopen Master Firmware is running as a thread in user mode. The CANopen Firmware benefits from all resources of the target platform, but is subjected to the restraints of the operating system kernel. Because the operating system kernel is not real time capable there are variations of the SYNC cycle time. To avoid system clock jitter, an external millisecond clock signal can be injected by the software to the CANopen Master API. It is not possible to execute the CANopen Firmware on the microcontroller of the CAN interface directly as with Windows VCI.

2.3 Function Categories

The CANopen Master API provides functions for the CANopen network management and the CANopen object communication in the following categories. For a detailed description see *Functions, p. 34*.

2.3.1 Basic API Functions

Basic API functions are used for initialization and parameterization of the API, to select the CAN board and for the communication with the Master firmware on the CAN board.

CAN Board Selection	
COP_InitBoard	Allocates a CAN interface for use by the CANopen Master API.
COP_ReleaseBoard	Cancels the interface/line combinations that are used by the CANopen Master API.
COP_GetBoardInfo	Requests information about the hardware properties of the interface in use and the version numbers of the software components.

Initialization and Parameterization of the API	
COP_InitInterface	Parameterizes the firmware on the CAN interface and defines the bitrate of the network and the node monitoring mechanism to be used by the CANopen Master firmware.
COP_DefineCallbacks	Declares functions of type `COP_t_EventCallback` from the Client application to the CANopen Master API.
COP_DefineMsgRPDO	Defines Windows messages that are posted to a window of the Client application in case of a new queue object.
COP_DefineMsgEvent	Defines Windows messages that are posted to a window of the Client application in case of a new queue object.
COP_DefineMsgEmergency	Defines Windows messages that are posted to a window of the Client application in case of a new queue object.
COP_DefineMsgSync	Defines Windows messages that are posted to a window of the Client application in case of a new queue object.
COP_GetThreadIds	Reads the thread identifiers of the internal CANopen Master API DLL poll threads for the receive queues.
COP_ClockTick1ms	Injects the main clock tick to the CANopen Master API in a millisecond interval.
COP_Reset_DLL	Deregisters all registered CAN interfaces and CAN lines and re-initializes the CANopen Master DLL.

Communication with the CANopen Master Firmware	
COP_SendMsg	For internal communication only, cannot be used for transmission to the bus
COP_GetMsg	For internal communication only, cannot be used for transmission to the bus
COP_SetCommTimeOut	For internal communication only, cannot be used for transmission to the bus
COP_GetStatus	Queries the status of the Master Firmware and the status of the DLL on the CAN interface.
COP_TestCommand	Tests if the firmware is correctly loaded onto the CAN interface and started.

2.3.2 Network Management Functions

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

Setting up the CANopen Network	
COP_AddNode	Registers a new node with the Master Firmware and adds the node to the network management.
COP_DeleteNode	Removes a node from the internal node list of the CANopen Master and from the network management.
COP_ImportEDS	Imports a CANopen device configuration file that allows an automated node configuration.
COP_SearchNode	Checks if a device with the specified node ID is available in the network.
COP_GetNodeInfo	Delivers the properties of a registered node.
COP_ChangeNodeParameter	Changes the properties of a registered node.
COP_SetEmcyIdentifier	Adapts the CAN identifier of the emergency object of a node.
COP_ConfigFlyMaster	Configures the Flying Master functionality of the Master Firmware.
COP_StartFlyMaster	Starts the Flying Master functionality of the Master Firmware.
COP_GetStatusFlyMasterNeg	Returns the current status of the negotiation of the network mastership with other potential Masters.

Controlling the Individual CANopen Nodes	
COP_StartNode	Sets a node or the network in state *operational*.
COP_StopNode	Sets a node or the network in state *stopped*.
COP_ResetComm	Resets the values of the communication profile of a node or the network.
COP_ResetNode	Resets the application and the values of the communication profile of a node or the network.
COP_EnterPreOperational	Sets a node or the network in state *pre-operational*.
COP_GetNodeState	Retrieves the current NMT state of a CANopen node.

2.3.3 CANopen Object Management Functions

The functions are used for creation and parametrization of CANopen communication objects.

Process Data Objects (PDO)	
COP_CreatePDO	Creates a PDO.
COP_DeletePDO	Removes a PDO and releases the resources.
COP_GetPDOInfo	Delivers the PDO properties.

Service Data Objects (SDO)	
COP_CreateSDO	Creates an additional Client SDO.
COP_GetSDOInfo	Delivers the properties of a Client SDO for SDO communication with a registered node.
COP_SetSDOTimeOut	Defines how long the CANopen Master Firmware waits for the individual response of the SDO server with a running SDO transfer.

System Services	
COP_DefSyncObj	Defines the cycle time for the system service of the synchronization object.
COP_SetSyncDivisor	Sets the divisor for the cycle time for the system service of the synchronization object of the Master Firmware.
COP_GetSyncInfo	Delivers the properties of the system service of the synchronization object.
COP_EnableSync	Starts the cyclic transmission of the synchronization object by the Master Firmware.
COP_DisableSync	Ends the cyclic transmission of the synchronization object by the Master Firmware.
COP_InitTimeStampObj	Transfers the current time for the system service or the central time information.
COP_StartStopTSObj	Starts and stops the cyclic transmission of the central time information.
COP_GetTimeStampObj	Delivers the properties and the current value of the system service of the central time information (Time Stamp Object).

2.3.4 CANopen Communication

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

Process Data Objects (PDO)	
COP_ReadPDO	Reads the data of a process data object (PDO) received by the Master Firmware from the RPDO queue.
COP_ReadPDO_S	Reads the data of a PDO received by the Master Firmware from the RPDO queue and returns the PDO data as a structure.
COP_RequestPDO	Initiates a request for a PDO.
COP_WritePDO	Write the data of a PDO to be transmitted by the Master Firmware into the TPDO queue.
COP_WritePDO_S	Write the data of a PDO to be transmitted by the Master Firmware into the TPDO queue. Accepts the PDO data as a structure.

Service Data Objects (SDO)	
COP_ReadSDO	Reads the contents of an object dictionary entry from a node.
COP_WriteSDO	Writes data into an object dictionary entry of a node
COP_PutSDO	Initiates reading or writing of a service data object (SDO) by placing an SDO operation in the transmit SDO queue.
COP_GetSDO	Reads the result of an SDO transfer from the receive SDO queue.
COP_ParallelPutSDO	Initiates reading or writing of a service data object (SDO) by placing an SDO operation in the transmit SDO queue.
COP_ParallelGetSDO	Reads the result of an SDO transfer from the receive SDO queue.
COP_CancelSDO	Cancels a running SDO operation.

System Services	
COP_GetEmergencyObj	Reads an emergency object from the EMCY queue and returns the emergency object subdivided into error value, error register and error data.
COP_GetEmergencyObj_S	Reads an emergency object from the EMCY queue and returns the alarm message as a structure.
COP_CheckSync	Checks whether a synchronization object is signaled by the Master Firmware.
COP_GetEvent	Reads a network or a Master Firmware event from the event queue.

2.3.5 LSS Services

The LSS services in accordance with CiA 305 *Layer Settings Services and Protocol (LSS)* are used to configure the parameters of the CANopen network for devices without a direct user interfaces (such as DIP switches). Not all devices support all LSS services.

COP_SetLSSTimeOut	Defines the delay time that determines how long a device response is awaited after transmitting an LSS command.
COP_LSS_InquireAddress	Reads the LSS address of a CANopen device.
COP_LSS_InquireNodeID	Inquires the node of a CANopen device.
COP_LSS_ConfigNodeID	Sets the node ID of the CANopen device via LSS services.
COP_LSS_ConfigBitTiming	Configures the bitrate of the CANopen device via LSS services.
COP_LSS_ActivateBitTiming	Switches the bitrate of all connected CANopen devices simultaneously via the LSS service.
COP_LSS_IdentifyRemoteSlaves	Identifies devices in the network, if the vendor ID and the product code are known.
COP_LSS_ IdentifyNonConfRemoteSlaves	Identifies whether devices exist in the network in the LSS Waiting State.
COP_LSS_Fastscan	Searches devices in the network that are in LSS Waiting State.

2.4 Exemplary Structure of a CANopen Master Application

The following figure shows a typical sequence of the CANopen Master API function called in a Windows application, including PDOs (process data objects) and SDOs (service data objects).

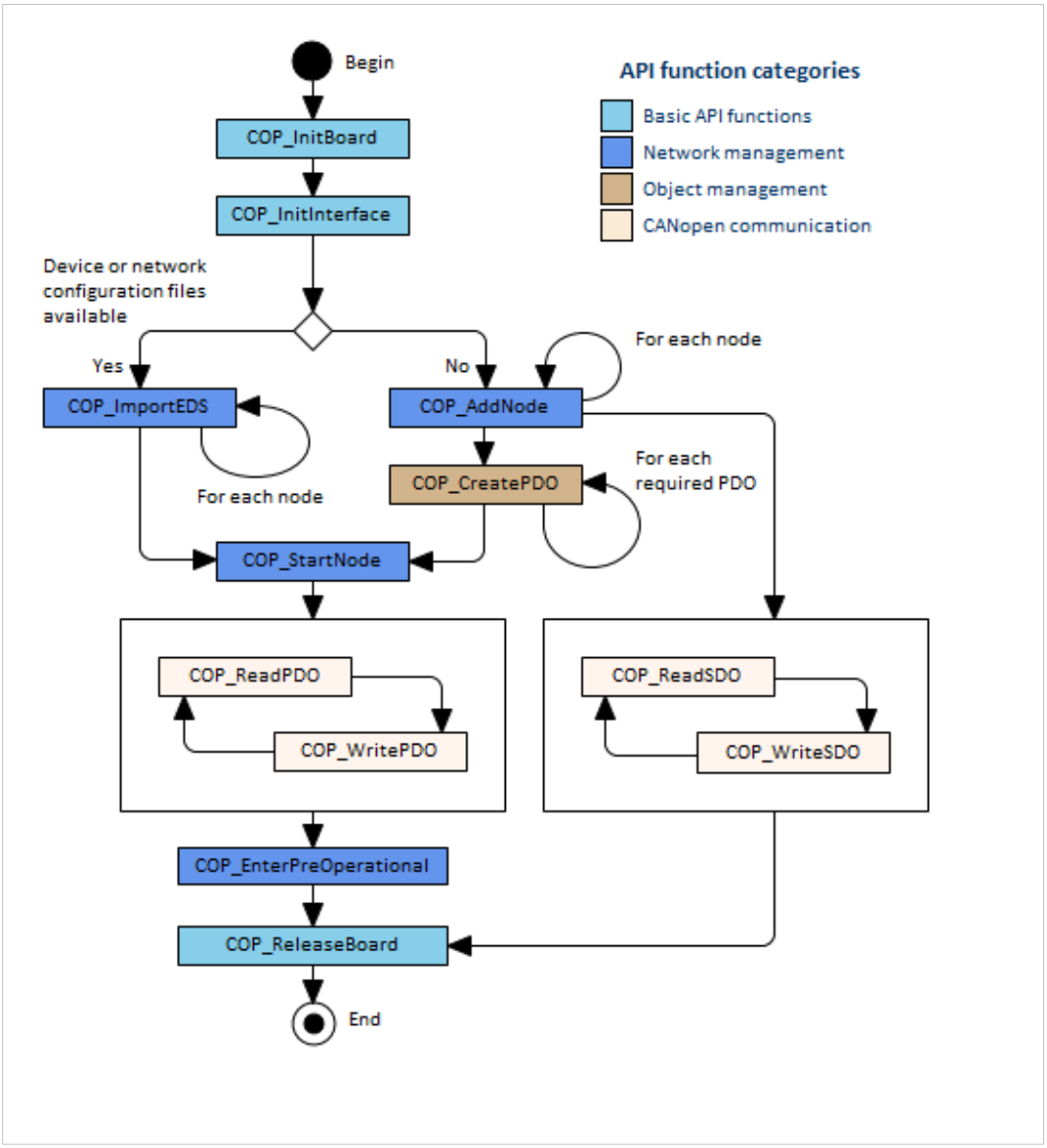

Fig. 2 Exemplary structure

3 Installation

3.1 System Requirements

The system requirements are mainly determined by the client application. The CANopen Master Firmware is running as thread in user mode of the Windows or Linux operating system and benefits from the internal clock of the system processor. The CANopen Master Firmware is also subject to the limitations of the operating kernel system. Since the operating system kernel is not real time capable there is increased jitter of the SYNC cycle time.

Recommended hardware configuration:

- The minimum recommended processor is a Pentium 4 processor with 1.3 GHz.
- The minimum main memory is 512 MB RAM.

3.1.1 Windows

- The software supports Windows XP/Vista/7/8/10 in 32 bit and 64 bit.
- The minimum required operating system is Windows XP Professional SP2.
- Installation of Ixxat VCI driver

> **!** Observe that the required VCI driver version is hardware dependent. Check the user manual of the hardware in use for information about required VCI driver version.

3.1.2 Linux

- The software is available for Linux Kernel 2.6.x, 3.x.x and 4.x.x in x86–64 architecture.
- Installation of either Ixxat SocketCAN driver or Ixxat ECI driver for Linux on x64 architecture version 1.1011

The ELF library has the following dependencies:

- `libdl.so.2`
- `libpthread.so.0`
- `libstdc++.so.6`
- `libm.so.6`
- `libqcc_s.so.1`
- `libc.so.6`

3.2 Installing the Software

> **!** Make sure, that the driver is installed before the CANopen Master API. Otherwise the error `BER_k_VCI_INST_ERR` is returned when calling the API function *COP_InitBoard*.

3.2.1 Windows

> **!** Make sure, that *XatCOP60_*.dll* is located in the same folder as the CANopen Master API DLL (*XatCOP60.dll/libXatCOP60.so*). Otherwise the error `BER_k_BOARD_DLL_ERR` is returned when calling the API function *COP_InitBoard*.

> **i** *The VCI versions VCI3 and VCI4 are supported by the CANopen Master API. Observe that the required VCI driver version is hardware dependent. Check the user manual of the hardware in use for information about required VCI driver version.*

- Download the VCI driver package from the support area on www.ixxat.com.
- Install the driver package.
- Make sure to have administrator rights.
- Start the file *setup63.exe* from the delivered CD.
- Follow the instructions of the program.
  - → By default, the CANopen Master API is installed in the directory *\Programs\Ixxat \CANopen Master API*.
  - → Sample applications are installed in *\Users\Public\Documents\Ixxat CANopen Master API*.
- Make sure, that *XatCOP60_VCI3.dll* is located in the same folder as the CANopen Master API DLL.

3.2.2 Linux

- Download the ECI driver package or the SocketCAN driver package from the support area on www.ixxat.com.
- Make sure to have superuser rights.
- Install the driver package.
- Unpack the zip file CANopen Master API for Linux from the delivered CD.
- To copy the two library files to */usr/lib* enter `sudo make install`.

4 Initialization

4.1 Initializing the CAN Interface

COP_InitBoard initializes the CAN interface and *COP_InitInterface* initializes the CANopen Master Firmware. The call of COP_InitBoard and COP_InitInterface forms a logical unit and the successful execution of the two functions is a condition for all further functions of the CANopen Master API. COP_InitInterface must not be called during operation.

COP_InitBoard:

- Specify the type of CAN interface in parameter *pBoardtype*.
- Specify the interface ID in parameter *pBoardID*.
- Specify the CAN line in parameter *ICANline*.
- Call the function COP_InitBoard.
  - → CAN interface is reset.
  - → Master firmware is loaded and started.
  - → Communication queues are created.
  - → Board handle *pBoardhdl* that identifies the CAN interface/line combination is returned.

If using a CAN interface with several CAN lines, it is possible to activate additional CAN lines. Call COP_InitBoard once for every CAN line to be used.

COP_InitInterface:

- Specify the CAN interface/line combination in parameter *boardhdl*.
- Specify the bitrate of the network in parameter *baudrate*.
- Specify the node ID of the CANopen Master firmware in parameter *node_no* (only relevant for use of heartbeat mechanism).
- Call the function COP_InitInterface.

4.2 Requesting Interface Information

- To request information about the hardware properties of the interface in use and the version numbers of the software components, call *COP_GetBoardInfo*.
- In parameter *boardhdl* specify the handle of interface/line combination.
  - → Buffer is provided by the Client application of data type COP_BOARD_INFO.
  - → COP_BOARD_INFO contains information about the properties of the interface in use and the version numbers of the software components.

4.3 Checking the Receive Queues

For the reception of CANopen communication objects, four separate receive queues exist.

Receive queue	Function to check the queue
PDO receive queue for process data objects	`COP_ReadPDO` in the PDO message handler
EMCY queue for emergency objects	`COP_GetEmergencyObj` in the emergency message handler
Event queue for network or firmware events	`COP_GetEvent` in the event message handler
SYNC queue for SYNC objects	`COP_CheckSync` in the SYNC message handler

The four receive queues must be read regularly with the corresponding function until the queue is empty, because their capacity is relatively small. Reading is possible either synchronously by polling (see *Polling, p. 19*) or asynchronously by callbacks (see *Callbacks, p. 19*). For more information and example code for callbacks see *Troubleshooting — Frequent Source of Errors, p. 126*

> **!** If the receive queues are not read regularly, each arriving CANopen object causes a queue overrun notification (`COP_GetEvent`).

4.3.1 Polling

RPDO queue

- Call *COP_ReadPDO* cyclically to check the receive queue.
- If an object is read from the queue, call `COP_ReadPDO` in a loop until the RPDO queue is empty.

For the EMCY queue, the event queue and the SYNC queue use the corresponding functions to check each queue.

4.3.2 Callbacks

If callbacks are defined, the callback functions are called by the CANopen Master API when a new receive object is entered in the data queue.

- To enable the CANopen Master API DLL to call back the application, implement the necessary callback functions of type *COP_t_EventCallback* in the client application.

With *COP_DefineCallbacks* the functions of type *COP_t_EventCallback* from the Client application are declared to the CANopen Mater API and called by the API in case of a new queue object.

- Specify the callback functions of type `COP_t_EventCallback` for each queue:
  - for the reception of a PDO in parameter *fp_rx_pdo*
  - for the reception of a EMCY object in parameter *fp_emergency*
  - for the reception of a network event in parameter *fp_net_event*
  - for the reception of a SYNC object in parameter *fp_sync*
- If no callback function is implemented, specify value **0** in the respective parameter.
  - → The implemented callback functions inform when a new receive object is entered in the corresponding data queue.
- Within the callback functions, call the corresponding function to check the queue until the receive queue is empty (*COP_ReadPDO*, *COP_GetEmergencyObj*, *COP_GetEvent*,*COP_CheckSync*).

It is possible to call `COP_DefineCallbacks` repeatedly while the program is running, for example to reregister callback functions temporarily, to re-register callback functions or to specify additional callback functions. Since the boardhdl identifies a specific CAN interface/line combination, different callback functions can be specified for the different CAN lines.

4.3.3 Messaging (Windows only)

Alternatively to the declaration of callback functions it is possible to specify messages to Windows and threads (PostMessage, PostThreadMessage) for each received queue entry with `COP_DefineMsgRPDO`, `COP_DefineMsgEvent`, `COP_DefineMsgEmergency`, `COP_DefineMsgSync`. If a message is defined, the message is transmitted to a window or a thread of the Client application with each new object in the respective queue. *wParam* of the messages contains the board handle and *lParam* contains the queue number.

5 Setting Up the CANopen Network

To set up the CANopen network existing device configuration files or existing network configuration files can be imported or nodes can be added via the API.

5.1 Importing Device Description and Configuration Files

The nodes can be configured by importing a CANopen device configuration file. The device description or configuration file contains PDOs, the Emergency object and node monitoring settings. The automated node configuration via a device description or configuration file overwrites the node monitoring settings via *COP_AddNode* and any other existing node configuration. By importing a device configuration file the CANopen Master API is adapted to the external node settings. The file extensions .eds, .dcf, and .cdc are accepted.

COP_ImportEDS supports the following files:

- device description files according to CiA 306
- device configuration files according to CiA 306
- concise device configuration files according to CiA 302-3

Import a configuration file with *COP_ImportEDS*:

- ► Specify the handle of the interface/line combination in parameter *boardhdl*.
- ► Specify the node ID of the external CANopen device in parameter *node_no*.
- ► Specify the filename, path and file format as zero terminated string in parameter *filename*.
- ► Call the function `COP_ImportEDS`.
- ► To check if the configuration is successful, call *COP_GetPDOInfo* or *COP_GetNodeInfo*.
- ► Import the configuration file for each desired node.
- ► Transfer the node in state *operational* with *COP_StartNode* (see *Starting the Network, p. 24*).

  → If configured, node monitoring is started.

To configure the network fully automated in one step, import a manager's concise device configuration file, that includes embedded node concise device configuration data, under the node_no of the Master API Firmware (as given in COP_InitInterface).

5.2 Adding Nodes and Creating PDOs

5.2.1 Adding Nodes

> **!** The number of nodes that can be managed depends on the CAN interface in use! Observe *Performance Characteristics, p. 106* for the exact value for different interfaces.

With `COP_AddNode` a new node is registered with the Master Firmware and added to the network management. The node registration is the condition for any communication with the corresponding CANopen device because it initializes internal management structures and status variables and establishes the first four receive and transmit PDOs according to the Predefined Connection Set.

`COP_AddNode`:

- Specify the handle of CAN interface/line combination in parameter *boardhdl*.
- Specify the node ID of the external CANopen device in parameter *node_no*.
- In parameter *NgOrHb* specify whether the node is monitored via node guarding or via heartbeat message. Mixed mode is also permitted.
- In case of heartbeat monitoring make sure that the client application configures the corresponding object dictionary entry [1017.00] (Producer Heartbeat Time) of the node appropriately.
- Specify the guard time for the heartbeat in milliseconds in parameter *GuardHeartbeatTime* (for more information see *Timer Resolutions and Value Ranges, p. 127*).
- Specify the permitted number of attempts of guard requests of the Master in *lifetimefactor* (only applies to node guarding mechanism, but not to heartbeat).
- Call the function for each desired node.

  → The first four receive PDOs and the first four transmit PDOs according to the Predefined Connection Set are established with data length 8 and transmission type `COP_k_PDO_MODE_ASYNC` (`COP_CreatePDO` does not have to be called for the first 8 PDOs).
- If required, create additional PDOs (see *Creating PDOs, p. 23*).
- Transfer the node in state *operational* with `COP_StartNode` (see *Starting the Network, p. 24*).

  → If configured, node monitoring is started.
- To check if the node exists in the network or to check the current state of a node, call `COP_SearchNode`.

5.2.2 Creating PDOs

> **!** The number of PDOs that can be handled simultaneously depends on the CAN interface in use! Observe *Performance Characteristics, p. 106* for the exact value of different interfaces.

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. The created local PDO is just mirroring an existing PDO on a physical node.

- Make sure the nodes are registered with the Master Firmware and added to the network management (see *Adding Nodes, p. 22*).

The first four transmit PDOs and the first four receive PDOs according to the Predefined Connection Set are created when calling *COP_AddNode* with data length 8 and transmission type `COP_k_PDO_MODE_ASYNC`.

COP_CreatePDO:

- Specify the handle of the CAN interface/line combination in parameter *boardhdl*.
- To allocate the PDO to a node, specify the node ID in the parameter *node_no* (valid values 0–127).

  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).
- Specify the number of the PDO in parameter *pdo_no* (valid values 1–16).
- Specify the transmission type in parameter *type*.
- Specify the PDO mode in parameter *mode*.
- Specify the length of the PDO in parameter *length* (valid values 1–8).
- Specify the identifier of the CAN object that is used by the PDO in parameter *CANid*.
- To alter the properties of a PDO after the network is started, call *COP_CreatePDO* again.
- Specify and call the function for each desired node.
- To read the properties of a created PDO, call function *COP_GetPDOInfo*.
- To delete a PDO and release the resources (e. g. the CAN identifier) call function *COP_DeletePDO*.

6 Controlling the Network

6.1 Starting the Network

With *COP_StartNode* transfer a node or the network in state *operational*:

- Specify the handle of the CAN interface/line combination in parameter *boardhdl*.
- To transfer individual nodes into *operational* state, specify the node in parameter *node_no*.

  or

- To transfer the network including all nodes into *operational* state, specify 0 in parameter *node_no*.

  → Specified nodes are transferred into *operational* state by the transmitting of an NMT command.

  → If configured, node monitoring is started.

- Query the current state of a network participant with *COP_GetNodeState*.

6.2 Requesting Information and Changing Settings

- To check if a device with a specified node ID is available in the network, call function *COP_SearchNode*.

  → While COP_SearchNode is running the command queue is blocked and no other API functions can be called.

- To get the properties of a registered node, call *COP_GetNodeInfo*.
- To get the current state of a network participant, call *COP_GetNodeState*.
- To change the settings of a registered node , call *COP_ChangeNodeParameter*.
- To adapt the CAN identifier of the Emergency object of a node, call *COP_SetEmcyIdentifier*.

6.3 Deleting a Node

- Stop the external node with *COP_StopNode* .

  or

- Reset the application and the values of the communication profile with *COP_ResetNode*.

  or

- Reset the values of the communication profile with *COP_ResetComm*.
- To delete the stopped or reset node from the internal node list of the CANopen Master and from the network management, call function *COP_DeleteNode*.

6.4 Activating the Flying Master Functionality

The Flying Master functionality ensures that there is always an active Master, even if the actual Master fails, according to CiA 302. The corresponding object dictionary entries are Flying Master Timing Parameters [1F90] and Consumer Heartbeat Time [1016] to monitor the active Master after the transfer of the network mastership.

- Observe, that the Flying Master functionality is only available with certain CAN interfaces (see *Performance Characteristics, p. 106*).
- Make sure that the interface is initialized with Flying Master functionality: `COP_InitInterface (AddFeatures = COP_k_FEATURE_FLYING_MASTER)`.
- To configure the Flying Master specify the input parameters of *COP_ConfigFlyMaster*.
- Observe, that the function can only be called once and therefore the configuration cannot be changed later.
- To start the Flying Master, call *COP_StartFlyMaster*.

  → CANopen Master Firmware actively participates in the negotiation of the active Master with other potential Masters and attempts to gain the network mastership based on its settings.
- To request the current status of the negotiation of the network mastership with other potential Masters, call *COP_GetStatusFlyMasterNeg*.

  → Status, node ID and priority class of the active Master are returned.

6.5 Stopping the Network

- Set the network into NMT state *Pre-operational* with *COP_EnterPreOperational*.

  or
- Set a node or the network to NMT state *Stopped* with *COP_StopNode*.
- If the application is closed, release each interface/line in use with *COP_ReleaseBoard* to reset the relevant CAN controller.

  → When each interface/line is released, the Master Firmware is unloaded and the interface is released for other applications.

7 Communicating with CANopen Devices

7.1 Process Data Objects (PDOs)

7.1.1 Reading PDOs

Only the four RPDOs of the Predefined Connection Set and the PDOs that are created with *COP_CreatePDO* can be received.

- Make sure, that the network is set up correctly (see *Setting Up the CANopen Network, p. 21*) and started (see *Starting the Network, p. 24*).

To read the data of PDOs that are received by the Master Firmware from the RPDO queue, call *COP_ReadPDO*:

- Specify the handle of the CAN interface/line combination in parameter *boardhdl*.
- Call the function `COP_ReadPDO`.
  - → The returned *node_no* and *pdo_no* serve as unique identifier.
  - → The number of valid data bytes is returned in *rxlen*.
- To query the properties of a PDO, call function *COP_GetPDOInfo*.

7.1.2 Writing PDOs

Only the four TPDOs of the Predefined Connection Set and the PDOs that are created with *COP_CreatePDO* can be transmitted.

- Make sure, that the network is set up correctly (see *Setting Up the CANopen Network, p. 21*) and started (see *Starting the Network, p. 24*).
- If necessary collect the communication parameters from the object dictionary with *COP_ReadSDO*.

To write the data of PDOs that are received by the Master Firmware to the TPDO queue, call *COP_WritePDO*:

- Specify the handle of the CAN interface/line combination in parameter *boardhdl*.
- Specify the node to which the PDO is to be transmitted in parameter *node_no*.
- Specify the number of the PDO (beginning with 1) in parameter *pdo_no*.
- Observe, that PDOs with incorrect parameters are rejected by the firmware but no error is returned, since `COP_WritePDO` works unconfirmedly.
- Specify the address of an 8 byte buffer for the PDO data to be transmitted in parameter *txdata*.
- Call the function `COP_WritePDO`.
- To query the properties of a PDO call function *COP_GetPDOInfo*.

Since the checking of the parameters *node_no* and *pdo_no* is done in the firmware, the function *COP_WritePDO* does not return an error code if the parameters are incorrect.

- To check the parameters *node_no* and *pdo_no*, read the event queue with *COP_GetEvent*.
  - → `COP_k_WPDO_EVT` is returned in case of invalid parameters.

7.2 Service Data Objects (SDOs)

7.2.1 Creating Additional Client SDOs

For each node, that is registered with *COP_AddNode* the default SDO exists. If a Server SDO object exists on a registered node, *COP_CreateSDO* creates an additional Client SDO in the internal Master Firmware object management. The appropriate Server SDOs has to be configured by the Client application via one of the respective object dictionary entries [1201]..[127F]. The CANopen Master API is always the SDO Client and the node is always the SDO Server.

- Create an additional Client SDO in the internal Master Firmware object management with *COP_CreateSDO*.
- Make sure, that the appropriate Server SDO is configured by the Client application.
- To get the properties of a Client SDO for SDO communication with a registered node, call function *COP_GetSDOInfo*.
- To change the properties, call function *COP_CreateSDO*.
- To change the default value (200 ms) for the waiting time of the CANopen Master Firmware for the response of the SDO Server with a running SDO transfer, call function *COP_SetSDOTimeOut*.

7.2.2 Reading and Writing SDOs

COP_ReadSDO reads the contents of an object dictionary entry of a node. *COP_WriteSDO* writes data into an object dictionary entry of a node. The functions work synchronously, therefore the call only returns to the Client application if the SDO transfer is finished or if the timeout is expired (default 200 ms, set with *COP_SetSDOTimeOut*).

To avoid blocking of the application when transferring larger amounts of data it is possible to read and write data asynchronously with *COP_PutSDO* and *COP_GetSDO*.

COP_WriteSDO and COP_ReadSDO (Synchronously)

Read the data of an object dictionary entry of a node with *COP_ReadSDO*:

- Address the object dictionary entry via the main index and the subindex in the parameters *idx* and *subidx*.
- Specify the number of the SDO to be used in parameter *sdo_no*:
  - To use the default SDO that is automatically created by the firmware, enter `COP_k_DEFAULT_SDO`.
  - To use an SDO created with *COP_CreateSDO*, enter `COP_k_USERDEFINED_SDO`.
- Specify the handle of the CAN interface/line combination in parameter *boardhdl*.
- Specify the node to read the object dictionary entry from in parameter *node_no*.
- Specify the SDO transmission protocol in parameter *mode*.

Example

To read the manufacturer ID (unsigned32 value) from the object dictionary of node 3, use the following command:

```
short COP_ReadSDO( hBoard, 3, COP_k_DEFAULT_SDO, COP_NO_BLOCKTRANSFER,
                   0x1018, 0x01, <rxlen>, <rxdata>, NULL)
```

Write data to an object dictionary entry of a node with *COP_WriteSDO*:

- Address the object dictionary entry via main index and subindex in the parameters *idx* and *subidx*.
- Specify the number of the SDO to be used in parameter *sdo_no*:
  - To use the default SDO that is automatically created by the firmware, enter `COP_k_DEFAULT_SDO`.
  - To use an SDO that is created with *COP_CreateSDO*, enter `COP_k_USERDEFINED_SDO`.
- Specify the handle of the CAN interface/line combination in parameter *boardhdl*.
- Specify the node to write the object dictionary entry to in parameter *node_no*.
- Specify the SDO transmission protocol in parameter *mode*.
- Specify the number of bytes to be transmitted in parameter *txlen*.
- Specify the address of the buffer that contains the object dictionary data in parameter *txdata*.

COP_PutSDO and COP_GetSDO (Asynchronously)

- To read and write SDOs asynchronously, place the SDO operation with `COP_PutSDO` in the transmit SDO queue.

  → Function returns immediately.
- Read the results of the terminated SDO transfer with `COP_GetSDO`.
- When calling `COP_GetSDO`, make sure, that the calling process and thread are the same as in the corresponding `COP_PutSDO` call.
- To abort an initiated SDO transfer, call *COP_CancelSDO*.

To abort all running SDO transfers, call `COP_CancelSDO` with parameter node_no=0 and parameter sdo_no=0.

The event handle that is transferred with `COP_PutSDO` can be used to wait for the end of the SDO transfer without polling with `COP_GetSDO`.

7.3 System Services

7.3.1 Synchronisation Object

- To read the properties of the synchronisation object, call function *COP_GetSyncInfo*.
- To change the default cycle time for the system service of the synchronisation object (1000 ms), call *COP_DefSyncObj*.

  → The transmitting of the synchronisation object is cancelled with calling *COP_DefSyncObj*.
- To implement different cycle times for different CAN lines set a divisor with *COP_SetSyncDivisor* (for more information see *Divisor for the Cycle Time, p. 29*).
- Start the transmitting of the synchronisation object with *COP_EnableSync*.

  → The configured value is applied to all CAN lines (the firmware only knows one time base).

  → The synchronisation queue is filled with a message with each call of the function `COP_EnableSync`.
- To avoid increased CPU load due to internal queue overruns, make sure that the synchronisation queue is read regularly with *COP_CheckSync* by the Client application.

  If a callback is assigned for the synchronisation queue, `COP_CheckSync` is automatically called when the Master transmits a synchronisation object.
- To stop the cyclic transmission of the synchronisation object, call function *COP_DisableSync*.

7.3.2 Divisor for the Cycle Time

To implement different cycle times for different CAN lines a divisor for the cycle time can be set with *COP_SetSyncDivisor*. The divisor defines at what internal cycle the SYNC object is transmitted and therefore allows to implement different cycle times for different CAN lines.

Example 1

- desired cycle time line 1: 5 ms
- desired cycle time line 2: 10 ms
- Set the greatest common divisor of the different cycle times in *COP_SetSyncDivisor* in parameter *sync_period* (5 ms).
- Divide the desired cycle time of line 1 (5 ms) by the greatest common divisor (5 ms), to get the divisor (1 ms).
- Set the divisor (2 ms) in *COP_SetSyncDivisor*.

  → In this example the firmware transmits every SYNC object on line 1.
- Divide the desired cycle time of line 2 (10 ms) by the greatest common divisor (5 ms), to get the divisor (2 ms).
- Set the divisor (2 ms) in *COP_SetSyncDivisor*.

  → In this example the firmware only transmits every second SYNC object on line 2.

Example 2

- desired cycle time line 1: 100 ms
- desired cycle time line 2: 30 ms
- greatest common divisor: 10 (*sync_period*)
- divisor for line 1: 100 ms / 10 ms = 10 (*divisor*)
- divisor for line 2: 30 ms / 10 ms = 3 (*divisor*)

7.3.3 Time Stamp Object

- Transfer the current time for the system service of the central time information (TimeStamp Object) with *COP_InitTimeStampObj*.
- Start the cyclic transmitting of the central time information by the Master Firmware with *COP_StartStopTSObj*.
- Observe information about the cycle time (parameter *cycle*) in *Timer Resolutions and Value Ranges, p. 127*.
- To read the properties and the current value of the time stamp object, call function *COP_GetTimeStampObj*.
- To change the properties call function `COP_InitTimeStampObj`.
- To stop the cyclic transmitting of the central time information by the Master Firmware, call function `COP_StartStopTSObj`.

7.3.4 Emergency Objects

- To read an emergency object from the EMCY queue, call function *COP_GetEmergencyObj*.

  → Emergency object is returned divided in error value, error register and error data.

8 Parameter Settings for Devices without User Interface (LSS Services)

With the LSS Services the parameters of the CANopen network for devices without direct user interface (such as DIP switches) can be set. The LSS Services are in accordance with CiA-305 *Layer Settings Services and Protocol*. Not all devices support LSS Services.

The worldwide unique identification of the device that supports LSS Services consists of the LSS address that corresponds with the identity object [1018].

Use functions *COP_LSS_InquireAddress*, *COP_LSS_InquireNodeID*, *COP_LSS_ConfigNodeID* and *COP_LSS_ConfigBitTiming* only in networks with **one** LSS compatible device. If more than one LSS compatible devices are present in the network, the device responses overlap and destroy each other (because of SwitchModeGlobal).

8.1 Setting the Node ID

- Specify the time to wait for the answer of the device with *COP_SetLSSTimeOut* (default: 100 ms).
- To read the LSS address of the device in networks with only one LSS compatible device, call *COP_LSS_InquireAddress* with the necessary parameters.

  → The following LSS command are issued:

  1. SwitchModeGlobal to activate the LSS Service
  2. InquireIdentiyVendorID to inquire the vendor identity *VendorId*
  3. InquireIdentityProductCode to inquire the product code *ProductCode*
  4. InquireIdentityRevisionNumber to inquire the revision number of the device *RevisionNo*
  5. InquireIdentitySerialNumber to inquire the serial number *SerialNo*
  6. SwitchModeGlobal to deactivate the LSS Service
- To read the node number of the device in networks with only one LSS compatible device, call *COP_LSS_InquireNodeID* with the necessary parameters.

  → The following LSS command are issued:

  1. SwitchModeGlobal to activate the LSS Service

     or

     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

  → Configured node ID is returned.

  → If no valid node ID is configured and the device is in *LSS Init State*, value 255 is returned.
- To configure the node number of the device in networks with only one LSS compatible device, call *COP_LSS_ConfigNodeID* with the necessary parameters.

- To set the device in *LSS Init State*, enter the value 255.

  → The following LSS command are issued:

  1. SwitchModeGlobal to activate the LSS Service

     or

     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
  4. SwitchModeGlobal to deactivate the LSS Service

  → Node ID is configured.

8.2 Setting the Baud Rate

- To configure the baud rate of the device in networks with only one LSS compatible device, call `COP_LSS_ConfigBitTiming` with the necessary parameters.

  → The following LSS command are issued:

  1. SwitchModeGlobal to activate the LSS Service

     or

     SwitchModeSelective to address the device using the parameters *VendorId*, *ProductCode*, *RevisionNo* and *SerialNo*
  2. ConfigureBitTimingParameters to configure the baud rate using the parameters *new_baudtable* and *new_baudrate*
  3. ActivateBitTimingParameters to directly activate the new baud rate after a delay time that is configured using the parameter *switch_delay*
  4. StoreConfiguration
  5. SwitchModeGlobal to deactivate the LSS Service

  → Baud rate is configured.

  → Firmware is reset internally.

- Initialize the CANopen Master Firmware (see *Initializing the CAN Interface, p. 18*).
- To switch the baud rate of all connected devices simultaneously, call `COP_LSS_ActivateBitTiming` with the necessary parameters.

  → The following LSS command are issued:

  1. SwitchModeGlobal to activate the LSS Service
  2. ActivateBitTimingParameters to directly activate the new baud rate after a delay time that is configured using the parameter *switch_delay*
  3. SwitchModeGlobal to deactivate the LSS Service

  → Firmware is reset internally.

- Initialize the CANopen Master Firmware (see *Initializing the CAN Interface, p. 18*).

8.3 Searching for Devices in the Network

If several devices of the same type exist in the network or if the serial number of a device is unknown, the device can be identified via *COP_LSS_IdentifyRemoteSlaves*. The device is identified via interactive narrowing of the serial number path.

- Specify time to wait for the answer of the device with *COP_SetLSSTimeOut* (default: 100 ms).
- To search for devices, call `COP_LSS_IdentifyRemoteSlaves`.
  - → The LSS command LSSIdentifyRemoteSlaves is issued using the parameters *VendorID*, *ProductCode*, *RevisionNoLow*, *RevisionNoHigh SerialNoLow* and *SerialNoHigh*.
  - → If at least one node of the specified serial number path responses within the specified waiting time, the function returns `COP_k_OK`.
- To check if devices in *LSS Waiting State* are present in the network, call *COP_LSS_IdentifyNonConfRemoteSlaves*.
  - → The LSS command LSS IdentfiyNonConfiguresRemoteSlaves is issued.
  - → If at least one node responses within the specified waiting time, the function returns `COP_k_OK`.

9 Functions

9.1 Basic API Functions

9.1.1 COP_InitBoard

Allocates a CAN interface for use by the CANopen Master API.

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

Parameter

Parameter	Dir.	Description
pBoardhdl	[out]	Identifies the CAN interface/line combination for all subsequent function calls.
pBoardtype	[in/out]	Type of the CAN interface, according to header files *vcguid* resp. *xatbrds* (supported interfaces see *Supported CAN Interfaces, p. 10*). Interfaces are described by the entry `GUID_AAAA_DEVICE` (e.g. `GUID_CANIB_PCIE_DEVICE` or `GUID_USB2CANV2_DEVICE`). Two special values are defined in the main header `COP`: `COP_DEFAULTBOARD`: Typical application if only one CAN interface is installed in the computer. The installed CAN interface is used. `COP_BOARDDIALOG` (Windows only): Dialog to select an interface is shown. The value is always returned at `COP_DEFAULTBOARD` and `COP_BOARDDIALOG` and can be saved in the persistent configuration data of the Client application.
pBoardID	[in/out]	Unique identifier of the CAN interface, used with `pBoardtype` to identify the interface locally. If only one interface of the selected type is present, enter value `COP_1stBOARD`. If several interfaces of the same `pBoardtype` are present, use consecutive `pBoardID` values (`COP_1stBOARD`, `COP_2ndBOARD`, `COP_3rdBOARD`, etc. are defined in the header (*Cop.h*). The value is always returned and can be saved (e.g. in the persistent configuration data of the Client application).
lCANline	[in]	Selects the CAN line to be used. The following default values are defined in the main header *Cop.h*: `COP_FIRSTLINE`: first CAN line, default value `COP_SECONDLINE`: second CAN line (if existing) `COP_THIRDLINE`: third CAN line (if existing) `COP_FOURTHLINE`: fourth CAN line (if existing) `COP_SINGLELINE`: single line firmware, a specially optimized and faster single line firmware is loaded onto the CAN interface, useful for applications where maximum performance is required with only one CAN line

Return Value

Return value	Description
BER_k_OK	Function succeeded
BER_k_ERR	General error, not further specified
BER_k_BOARD_ALREADY_USED	Interface already used by CANopen Master API
BER_k_ALL_BOARDS_USED	Maximum capacity of 4 simultaneously operable interfaces already in use
BER_k_CANNOT_SEARCH_BOARD	No interface selected, because Ixxat hardware selection dialog was cancelled
BER_k_BOARD_NOT_FOUND	Specified interface type and key do not match any available interface
BER_k_BOARD_NOT_SUPP	Specified interface is not supported by CANopen Master API due to unsuitable microcontroller or memory extension
BER_k_WRONG_FW	Version number supplied by the firmware is unsuitable, indicates faulty communication between PC and µC
BER_k_USED_FROM_OTHER_PROCESS	Specified interface already occupied by another CAN application
BER_k_PC_MC_COMM_ERR	Communication set up with interface not successful
BER_k_BOARD_DLD_ERR	Error occurred during firmware download. Possible reason: the generic VCI3 firmware library *XatCOP_VCI3.dll* is missing or can not be downloaded. Make sure that firmware library is located in the same folder as main library *XatCOP60.dll*. Also indicates that I/O address range is used by another hardware or driver.
BER_k_NO_SUCH_CANLINE	Specified CAN line does not exist or is not supported by the firmware.
BER_k_CANLINE_USED	Specified CAN line already in use
BER_k_VCI_INST_ERR	Basic VCI driver not available or defective
BER_k_BOARD_ERR	Incorrect or unknown interface type
BER_k_CCI_INST_ERR	CCI installation error (internal)
BER_k_SDO_INST_ERR	Error while instancing or configuring SDO handler (internal)

Remark

Call the function once for every CAN line. For more information see *Initializing the CAN Interface, p. 18*.

9.1.2 COP_ReleaseBoard

Cancels the interface/line combinations that are used by the CANopen Master API. The interface is released when all combinations are closed. `COP_ReleaseBoard` must be called for each interface/line combination that is used by *`COP_InitBoard`* to reset the CAN controller.

```
void COP_ReleaseBoard( COP_t_HANDLE boardhdl )
```

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination

Return Values

No return values.

9.1.3 COP_GetBoardInfo

Requests information about the hardware properties of the interface in use and the version numbers of the software components.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
sp_info	[out]	Pointer to a buffer of data type COP_BOARD_INFO, that records the information (provided by the Client application)

COP_BOARD_INFO

Field	Type	Description
hw_version	WORD	Revision number of CAN interface (e. g. 0x0101 is V 1.01)
fw_version	WORD	Version number of Master Firmware (e. g. 0x0640 is V 6.40)
sw_version	WORD	Version number of CANopen Master API (e. g. 0x0600 is V 6.00)
board_seg	WORD	I/O address of interface in use, legacy element, only used if generic VCI firmware is running, then value 0x100 is returned.
irq_num	WORD	Interrupt request line IRQ used by the interface, legacy element, not used with Master API 6
canlines	WORD	Number of supported CAN lines
serial_num[16]	char[]	Serial number of the CAN interface
str_hw_type[40]	char[]	Description of the card type

Return Values

Return value	Description
BER_k_OK	Function succeeded
BER_k_ERR	Invalid handle
COP_k_IV	NULL pointer as parameter

9.1.4 COP_InitInterface

Parameterizes the firmware on the CAN interface and defines the bitrate of the network and the node monitoring mechanism to be used by the CANopen Master firmware. The successful initialization by COP_InitInterface and *COP_InitBoard* is a condition for all further functions.

> **!** Do not call during operation!

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
baudtable	[in]	Number of bitrate table to be used. Two different tables are possible: **COP_k_BAUD_CIA**: Baud rates specified by CiA 301, standard table **COP_k BAUD_USER**: User defined bit timing values, depends on hardware if supported, for more information see header file *CopUserBittiming.h*
baudrate	[in]	Predefined baud rates of both tables. The following values are valid: 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 (valid values 0–127, standard value 0). A node ID (1–127) is only necessary if the CANopen network is operated with heartbeat node monitoring mechanism. The health of the CANopen Master Firmware can then be monitored by other subscribers.
hbTime	[in]	Heartbeat interval of the CANopen Master Firmware in milliseconds, 0 is deactivated (valid values 5–32767).
AddFeatures	[in]	Activate additional functionality. Three predefined values are possible: **COP_k_NO_FEATURES**: Default value, no additional functionality. **COP_k_FEATURE_FLYING_MASTER**: Activates the Flying Master functionality according to CiA-302. Not supported by all interface types **COP_k_FEATURE_LOWSPEED**: Activates low-speed bus coupling instead of the default high-speed coupling. Baud rate is limited to 125 kB (COP_k_125_KB) with low-speed.

Return Value

Return value	Description
COP_k_OK	Function succeeded
COP_k_CAL_ERR	General error of Master Firmware
COP_k_IV	Invalid parameter value
BER_k_ERR	Invalid handle
BER_k_NOT_SENT	Operation not entered in transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
BER_k_CANLINE_USED	CAN line already initialized
COP_k_NO_FLY_MASTER_PRESENT	Flying Master functionality not supported
COP_k_NO_LOWSPEED	Low-speed bus coupling not present or not supported

Remark

For more information about the value range and the resolution of *hbTime* see *Timer Resolutions and Value Ranges, p. 127*.

9.1.5 COP_DefineCallbacks

Declares functions of type `COP_t_EventCallback` from the Client application to the CANopen Master API. The declared functions are called by the API in case of a new queue object.

```
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 );
```

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
fp_rx_pdo	[in]	Called when a PDO is received, see *COP_ReadPDO*
fp_emergency	[in]	Called when an emergency object (EMCY) is received, see *COP_GetEmergencyObj*
fp_net_event	[in]	Called when a network event occurs, see *COP_GetEvent*
fp_sync	[in]	Called when a self-transmitted SYNC object is received, see *COP_CheckSync*

Return Value

Return value	Description
`COP_k_OK`	Function succeeded
`BER_k_ERR`	Invalid handle
`BER_k_BADCALLBACK_PTR`	Invalid function pointer

Remark

It is possible to call the function repeatedly while the program is running, for example to deregister callback functions temporarily, to re-register callback functions or to define additional callback functions. Since the *boardhdl* identifies a specific CAN interface/line combination, different callback functions can be defined for the different CAN lines. To deregister a callback function, use value 0 in the corresponding parameter.

9.1.6 COP_t_EventCallback

`COP_t_EventCallback` is a function prototype for functions within the Client application, that are registered with the CANopen Master API with *`COP_DefineCallbacks`* and called for signalling of a new object in a receive data queue.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination

que_num	[in]	Contains the queue ID, the last digit of the constants shows the CAN line, possible values:
		`COP_M2P_QUEUE_PDO0` resp. `COP_M2P_QUEUE_PDO1`
		`COP_M2P_QUEUE_EMERGENCY0` resp. `COP_M2P_QUEUE_EMERGENCY1`
		`COP_M2P_QUEUE_EVENT0` resp. `COP_M2P_QUEUE_EVENT1`
		`COP_M2P_QUEUE_SYNC0` resp. `COP_M2P_QUEUE_SYNC1`

canline	[in]	CAN line on that the new object is received

Return Value

No return values

9.1.7 COP_DefineMsgRPDO, COP_DefineMsgEvent, COP_DefineMsgEmergency, COP_DefineMsgSync

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. These functions are only available on Windows.

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

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

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

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
hWnd	[in]	Handle of the window to which the in *Msg* defined message is posted, value 0 deactivates the message posting
idThread	[in]	Identifier of the thread to which the in *Msg* defined message is posted, value 0 deactivates the message posting
Msg	[in]	When an object of the respective queue is received, this message is posted.

Return Value

Return value	Description
COP_k_OK	Function succeeded
BER_k_ERR	Invalid handle
COP_k_IV	Invalid window handle

Remark

It is possible to call the functions repeatedly while the program is running, for example to deactivate or to change messages temporarily or to change from windows messages to thread messages. Since the *boardhdl* identifies a specific CAN interface/line combination, different messages can be defined for the different CAN lines.

9.1.8 COP_GetThreadIds

Reads the thread identifiers of the internal CANopen Master API DLL poll threads for the receive queues.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/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 Value

Return value	Description
`BER_k_OK`	Function succeeded
`BER_k_CCI_INST_ERR`	Incomplete or defect interface initialization
`BER_k_ERR`	Invalid handle

Remark

To avoid instability of the Master API DLL resp. of the Client application, do not use the `OpenThread()` function to access the poll thread in the Master API DLL via the Client application.

9.1.9 COP_ClockTick1ms

Injects the main clock tick to the CANopen Master API in a millisecond interval.

```
short COP_ClockTick1ms( COP_t_HANDLE boardhdl,
                   const COP_e_CLOCKSOURCE clocksource );
enum COP_e_CLOCKSOURCE {
     COP_CLOCKSOURCE_EXTERNAL,
     COP_CLOCKSOURCE_INTERNAL };
```

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
clocksource	[in]	Switch between internal and external clock tick. `COP_CLOCKSOURCE_EXTERNAL`: shuts down the internal clock generator when called the first time. `COP_ClockTick1ms(COP_ CLOCKSOURCE_ EXTERNAL)` needs to be called every millisecond by the application as long as it is running. `COP_CLOCKSOURCE_INTERNAL`: reactivates the internal clock generator. Calling `COP_ClockTick1ms()` periodically is not necessary any more.

Return Value

Return value	Description
BER_k_OK	Function succeeded
BER_k_CCI_INST_ERR	Incomplete or defect interface initialization
BER_k_BOARD_NOT_SUPP	Firmware clock tick cannot be replaced for the CAN board in use
BER_k_ERR	Invalid handle
COP_k_IV	Invalid clock source switch

Remark

The CANopen firmware relies on a periodical millisecond clock tick. Typically, the clock tick is generated internally, but it can be replaced by an externally generated clock tick in case a high precision (hardware) clock signal is both available and necessary for the application. The clock tick is required to generate the Sync object, to handle node monitoring, and to measure SDO and LSS timeout. Calling `COP_ClockTick1ms()` each millisecond serves as a replacement clock tick for CANopen firmware.

9.1.10 COP_Reset_DLL

Deregisters all registered CAN interfaces and CAN lines and re-initializes the CANopen Master DLL.

```
void COP_Reset_DLL();
```

Parameter

No parameters

Return Value

No return values

9.1.11 COP_SendMsg

HMS recommends not to implement direct calls to `COP_SendMsg` because this API function will be removed with the next revision of the CANopen Master API.

Used internally for communication between CANopen Master API DLL and firmware. Writes an entry of type `COP_t_Message` to the transmit command queue.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
sp_message	[in]	Pointer to the operation record for the Master Firmware

Return Value

Return value	Description
`COP_k_OK`	Function succeeded
`BER_k_NOT_SENT`	Operation is not entered in the transmit command queue.
`BER_k_ERR`	Invalid handle

9.1.12 COP_GetMsg

HMS recommends not to implement direct calls to `COP_GetMsg` because this API function will be removed with the next revision of the CANopen Master API.

Used internally for communication between CANopen Master API DLL and firmware. Reads an entry of type `COP_t_Message` from the receive command queue.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
sp_message	[in]	Pointer to the confirmation record for the response of the Master Firmware

Return Value

Return value	Description
`COP_k_OK`	Function succeeded
`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, transmission path from Windows DLL to device firmware failed (USB, Ethernet…)
`BER_k_ERR`	Invalid handle

9.1.13 COP_SetCommTimeOut

HMS recommends not to implement `COP_SetCommTimeOut` because this structure will be removed with the next revision of the CANopen Master API.

Defines the waiting time for the acknowledgement of the Master Firmware.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
w_timeout	[in]	Delay time in milliseconds, valid values 55–65535, lower values are rounded up internally

Return Value

Return value	Description
`BER_k_OK`	Function succeeded
`BER_k_ERR`	Invalid handle

Remark

The communication waiting time is internally coupled to the SDO timeout. If this time span is set to less than the SDO timeout, the SDO timeout is automatically reduced.

9.1.14 COP_GetStatus

Queries the status of the Master Firmware and the status of the DLL on the CAN interface.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
state_master	[out]	Status of the CANopen Master Firmware, possible values: `COP_k_INIT`: Master is parameterized and ready for use (function *`COP_InitInterface`* is already called) `COP_k_NOT_INIT`: Master not yet parameterized by *`COP_InitInterface`*
state_err_dll	[out]	Status of the DLL of the Master Firmware, possible values: `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 status bit set `COP_k_DLL_ERESET`: CAN controller error status bit set

Return Value

Return value	Description
BER_k_OK	Function succeeded
BER_k_ERR	Invalid handle
BER_k_NOT_SENT	Operation not entered in the transmit command queue
BER_k_TIMEOUT	No response from Master Firmware

Remark

A change of the status of the DLL is also signaled via the event queue and can be read with *COP_GetEvent*.

9.1.15 COP_TestCommand

Tests if the firmware is correctly loaded onto the CAN interface and started. The requested and checked test string also checks if the communication between CANopen Master API DLL and the Master Firmware is working correctly via both command queues.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination

Return Value

Return value	Description
COP_k_OK	Firmware started, communication working
BER_k_ERR	Invalid handle
BER_k_NOT_SENT	Operation not 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

9.2 Network Management

9.2.1 COP_AddNode

Registers a new node with the Master Firmware and adds the node to the network management. The node registration is the condition for any communication with the corresponding CANopen device, because it initializes internal management structures and status variables and establishes the first four receive and transmit PDOs, according to the Predefined Connection Set.

> **!** The number of nodes that can be managed depends on the CAN interface in use! Observe *Performance Characteristics, p. 106* for the exact value of different interfaces.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
node_no	[in]	Node ID of the external CANopen device (valid values 1–127)
NgOrHb	[in]	Set monitoring mechanism for the node, possible values: **COP_k_NODE_GUARDING**: Node is guarded (the Master cyclically requests a predefined toggling CAN message) **COP_k_HEARTBEAT**: Node cyclically transmits heartbeat message (the node transmits a predefined CAN message on own initiative)
GuardHeartbeatTime	[in]	Define the guard time in milliseconds. With COP_k_NODE-GUARDING the parameter determines the time period between two guarding request messages (guard time). 0 means not guarded. With COP_k_HEARTBEAT the parameter determines the time period between two heartbeat messages (heartbeattime).
lifetimefactor	[in]	Define the number of permitted unsuccessful attempts at a guard request by the Master (valid values 1–255). If exceeded without a response of the node the firmware signals a COP_k_NMT_EVT in the event queue. Only for COP_k_NODE_GUARDING. For COP_k_HEARTBEAT the parameter is ignored.

Return Value

Return value	Description
BER_k_ERR	Invalid handle
BER_k_NOT_SENT	Operation not entered in transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Function succeeded
COP_k_CAL_ERR	Required CAN resource not available
COP_k_IV	Invalid parameter value

Remark

Depending on the CAN interface in use the number of nodes that can be managed simultaneously varies. For more information see *Performance Characteristics, p. 106*.

9.2.2 COP_DeleteNode

Removes a node from the internal node list of the CANopen Master and from the network management. Ensure, that the external node is stopped or reset with NMT commands before deleting.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
node_*no*	[in]	Node ID of the external CANopen device, valid values 1–127

Return Value

Return value	Description
COP_k_OK	Function succeeded
BER_k_ERR	Invalid handle
BER_k_NOT_SENT	Operation not entered in the transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_NOT_FOUND	No node registered with the stated node ID
COP_k_IV	Invalid node ID

9.2.3 COP_ImportEDS

Imports a CANopen device configuration file that allows an automated node configuration. The automated node configuration via a configuration file overwrites the node monitoring settings and any other existing node configuration. The file formats .EDS, .DCF and .CDC are supported. By importing a configuration file the CANopen Master API is adapted to the external node settings.

```
short COP_ImportEDS( COP_t_HANDLE boardhdl,
                     BYTE         node_no,
                     wchar_t*     filename,
                     DWORD        rules,
                     WORD*        idx,
                     BYTE*        subidx );
```

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
node_no	[in]	Node ID of the external CANopen device, valid values 1–127
filename	[in]	Absolute file name and path of the CANopen device description file as zero terminated string, .EDS, .DCF or .CDC
rules	[in]	Has to be initialized by 0, reserved as flag field to control the behavior of the function
idx	[out]	Optional parameter: in case of a configuration error, the OD entry index of the device configuration file that broke the import is delivered. If the return value is `COP_k_OK` the function continued despite the erroneous OD entry.
subidx	[out]	Optional parameter: in case of a configuration error, the OD entry subindex of the device configuration file that broke the import is delivered. If the return value is `COP_k_OK` the function continued despite the erroneous OD entry.

Return Value

Return value	Description
`COP_k_OK`	Function succeeded
`COP_k_IV`	Invalid parameter
`BER_k_ERR`	Invalid handle
`BER_k_NOT_SENT`	Operation not entered in the transmit command queue
`BER_k_TIMEOUT`	No response from Master Firmware
`BER_k_EDS_FILENOTFOUND`	Invalid parameter `filename`, device description file not found
`BER_k_EDS_CORRUPT`	Fatal error during parsing of device description file, file import failed, node configuration failed
`BER_k_EDSLIB`	External library *EDSLIB.dll* not found (only required for EDS/DCF device description import)

Remark

To check the result of the automated node configuration the Client application can call *COP_GetPDOInfo* and *COP_GetNodeInfo*.

To configure the network fully automated in one step, import a manager's CDC file, that includes embedded node CDC data, under the node_no of the Master API Firmware (as given in COP_InitInterface).

9.2.4 COP_SearchNode

Checks if a device with the specified node ID is available in the network. For the check the mandatory object dictionary entry [1000] is accessed via the standard Server SDO of the node. The timeout is 100 ms (independent of the SDO timeout). `COP_SearchNode` works synchronous and therefore blocks the command queue while running (no other COP_ functions can be called).

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
node_no	[in]	Node ID of the CANopen device searched for, valid values 1–127

Return Value

Return value	Description
COP_k_OK	Function succeeded
BER_k_ERR	Invalid handle
BER_k_NOT_SENT	Operation not entered in the transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_NOT_FOUND	No node registered with the stated node ID
COP_k_TIMEOUT	No node with stated node ID found in the network
COP_k_SDO_RUNNING	Default SDO channel is already used by another SDO transfer. Retry later.
COP_k_IV	Invalid node ID
COP_k_BSY	SDO engine is working to capacity. Retry later.

9.2.5 COP_GetNodeInfo

Delivers the properties of a registered node.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
node_no	[in]	Node ID of the CANopen device searched for, valid values 1–127
NgOrHb	[out]	Returns the monitoring mechanism for the node: **COP_k_NODE_GUARDING**: Node is guarded (the Master cyclically requests a predefined toggling CAN message) **COP_k_HEARTBEAT**: Node cyclically transmits heartbeat message (the node transmits a predefined CAN message on own initiative)
GuardHeartbeatTime	[out]	Returns the guard time in milliseconds. With COP_k_NODE-GUARDING the parameter returns the time period between two guarding request messages (guard time). 0 means not guarded. With COP_k_HEARTBEAT the parameter returns the time period between two heartbeat messages (heartbeattime).
lifetimefactor	[out]	Returns the number of permitted unsuccessful attempts at a guard request by the Master (valid values 1–255). If exceeded without a response of the node the firmware signals a COP_k_NMT_EVT in the event queue. Only for COP_k_NODE_GUARDING.
EmcyIdentifier	[out]	Returns the CAN identifier of the emergency object of the node. According to the Predefined Connection Set the default value is 0x80+node_no. This may be changed by overwriting OD entry [1014] of the node. Master Firmware must follow by calling *COP_SetEmcyIdentifier*.

Return Value

Return value	Description
COP_k_OK	Function succeeded
BER_k_ERR	Invalid handle
BER_k_NOT_SENT	Operation not entered in the transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_NOT_FOUND	No node registered with the stated node ID
COP_k_IV	Invalid node ID

9.2.6 COP_ChangeNodeParameter

Changes the properties of a registered node.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
node_no	[in]	Node ID of the external CANopen device (valid values 1–127)
NgOrHb	[in]	Set monitoring mechanism for the node: **COP_k_NODE_GUARDING**: Node is guarded (the Master cyclically requests a predefined toggling CAN message) **COP_k_HEARTBEAT**: Node cyclically transmits heartbeat message (the node transmits a predefined CAN message on own initiative)
GuardHeartbeatTime	[in]	Define the guard time in milliseconds. With COP_k_NODE-GUARDING the parameter determines the time period between two guarding request messages (guard time). 0 means not guarded. With COP_k_HEARTBEAT the parameter determines the time period between two heartbeat messages (heartbeattime).
lifetimefactor	[in]	Define the number of permitted unsuccessful attempts at a guard request by the Master (valid values 1–255). If exceeded without a response of the node the firmware signals a COP_k_NMT_EVT in the event queue. Only for COP_k_NODE_GUARDING.
EmcyIdentifier	[in]	Set the CAN identifier of the emergency object of the node. According to the Predefined Connection Set the default value is 0x80+node_no. This may be changed by overwriting OD entry [1014] of the node. Master Firmware must follow by calling *COP_SetEmcyIdentifier*.

Return Value

Return value	Description
BER_k_ERR	Invalid handle
BER_k_NOT_SENT	Operation not entered in transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Function succeeded
COP_k_NOT_FOUND	No node registered with stated node ID
COP_k_IV	Invalid parameter value

Remark

For information about the value range and the resolution of the GuardHeartbeatTime see *Timer Resolutions and Value Ranges, p. 127*.

9.2.7 COP_SetEmcyIdentifier

Adapts the CAN identifier of the emergency object of a node.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
node_no	[in]	Node ID of the external CANopen device, valid values 1–127
EmcyIdentifier	[out]	New CAN identifier of the emergency object of the node. The standardized CAN identifier is one of the highest priorities of CANopen. The CAN identifier is calculated from the node ID with the following formula: EmcyIdentifer = 0x80 +node_no.

Return Value

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

Remark

Each network participant transmits its emergency object on a reserved individual CAN identifier which is calculated in a standardized way according to the node ID. Since the emergency identifier can be reconfigured by writing to object dictionary entry [1014], the CANopen Master API allows to adapt to a reconfiguration with this function.

9.2.8 COP_ConfigFlyMaster

Configures the Flying Master functionality of the Master Firmware according to CiA-302. The corresponding object dictionary entries are Flying Master Timing Parameters [1F90] and Consumer Heartbeat Time [1016] to monitor the active Master after the transfer of the network mastership. The Flying Master functionality must be activated with the initialization of the firmware (see *COP_InitInterface*). Can only be called once.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
wDetectionTimeout	[in]	Delay time until the detection of an active network Master, corresponds to object directory entry [1F90sub1]
wNegotiationDelay	[in]	Delay time until negotiation of the network mastership among the Master candidates, corresponds to object directory entry [1F90sub2]
wPriorityLevel	[in]	Priority level of CANopen Master API, corresponds to object directory entry [1F90sub3], valid values: 0 (high), 1 and 2 (low)
wPriorityTimeslot	[in]	Object directory entry [1F90sub4]
wNodeTimeslot	[in]	Object directory entry [1F90sub5]
wCycletimeCd	[in]	Object directory entry [1F90sub6]
wCycletimeTimeoutH-beat	[in]	Monitoring of the active Master after transfer of network mastership, corresponds to directory entry [1016]

Return Values

Return value	Description
BER_k_ERR	Invalid handle
BER_k_NOT_SENT	Operation not entered in transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Function succeeded
COP_k_IV	Invalid 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

Remark

For information about CAN interfaces that support the Flying Master functionality see *Performance Characteristics, p. 106*.

9.2.9 COP_StartFlyMaster

Starts the Flying Master functionality of the Master Firmware. After starting the CANopen Master Firmware actively participates in the negotiation of the active Master with other potential Masters and attempts to gain network mastership based on its settings. The Flying Master functionality must be activated with the initialization of the firmware (see *COP_InitInterface*) and configured with *COP_ConfigFlyMaster*.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination

Return Value

Return value	Description
BER_k_ERR	Invalid handle
BER_k_NOT_SENT	Operation not entered in transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Function succeeded
COP_k_IV	Unauthorized parameter value
COP_k_UNKNOWN	Function already successful called
COP_k_NO_FLY_MASTER_PRESENT	Flying Master functionality not activated

9.2.10 COP_GetStatusFlyMasterNeg

Returns the current status of the negotiation of the network mastership with other potential Masters. Flying Master functionality must be activated with the initialization of the firmware (see *COP_InitInterface*), configured with *COP_ConfigFlyMaster* and started with *COP_StartFlyMaster*. The function can be called regularly to check whether the CANopen Master Firmware has network mastership.

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

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
status	[out]	Status of negotiation, possible values: **COP_k_FLY_MASTER**: CANopen Master Firmware has network mastership and is the active NMT Master. **COP_k_FLY_NOT_MASTER**: CANopen Master Firmware lost 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**: Negotiation is running and not yet concluded.
masterid	[out]	Node ID of active NMT Master if it is not CANopen Master Firmware
masterprio	[out]	Priority class of active NMT Master if it is not CANopen Master Firmware

Possible Return Values

Return value	Description
BER_k_ERR	Invalid handle
BER_k_NOT_SENT	Operation not entered in transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Function succeeded
COP_k_IV	Unauthorized parameter value
COP_k_UNKNOWN	Function already successful called
COP_k_No_FLY_MASTER_PRESENT	Flying Master functionality not activated

9.2.11 COP_StartNode

Sets a node or the network in state *operational* by transmitting an NMT command.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
node_no	[in]	Node ID of the external CANopen device, valid values 0–127 (0 means all nodes)

Return Value

Return value	Description
BER_k_ERR	Invalid handle
BER_k_NOT_SENT	Operation not entered in transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Function succeeded
COP_k_NOT_FOUND	No node registered with stated node ID
COP_k_IV	Invalid node ID

9.2.12 COP_StopNode

Sets a node or the network in state *stopped* by transmitting an NMT command.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
node_no	[in]	Node ID of the external CANopen device, valid values 0–127 (0 means all nodes)

Return Value

Return value	Description
BER_k_ERR	Invalid handle
BER_k_NOT_SENT	Operation not entered in transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Function succeeded
COP_k_NOT_FOUND	No node registered with stated node ID
COP_k_IV	Invalid node ID

9.2.13 COP_ResetComm

Resets the values of the communication profile of a node or the network by transmitting an NMT command.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
node_no	[in]	Node ID of the external CANopen device, valid values 0–127 (0 means all nodes)

Return Value

Return value	Description
BER_k_ERR	Invalid handle
BER_k_NOT_SENT	Operation not entered in transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Function succeeded
COP_k_NOT_FOUND	No node registered with stated node ID
COP_k_IV	Invalid node ID

9.2.14 COP_ResetNode

Resets the application and the values of the communication profile of a node or of the network by transmitting an NMT command.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
node_no	[in]	Node ID of the external CANopen device, valid values 0–127 (0 means all nodes)

Return Value

Return value	Description
BER_k_ERR	Invalid handle
BER_k_NOT_SENT	Operation not entered in transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Function succeeded
COP_k_NOT_FOUND	No node registered with stated node ID
COP_k_IV	Invalid node ID

9.2.15 COP_EnterPreOperational

Sets a node or the network in state *pre-operational* by transmitting an NMT command.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
node_no	[in]	Node ID of the node to be controlled, valid values 0–127 (0 means all nodes)

Return Value

Return value	Description
BER_k_ERR	Invalid handle
BER_k_NOT_SENT	Operation not entered in transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Function succeeded
COP_k_NOT_FOUND	No node registered with stated node ID
COP_k_IV	Invalid node ID

9.2.16 COP_GetNodeState

Retrieves the current NMT state of a CANopen node.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
node_no	[in]	Node ID of the desired node, valid values 0–127 (0 means all nodes)
node_state	[out]	State of the network participant, possible values: **COP_k_NS_STOPPED**: node in state *stopped* **COP_k_NS_OPERATIONAL**: node in state *operational* **COP_k_NS_PREOPERATIONAL**: node in state *pre-operational*. **COP_k_NS_UNKNOWN**: node state is unknown or node is not registered with Master Firmware. Possible reason: because of missing node monitoring or turned off heartbeat the NMT state can not be deduced. **COP_k_NS_DISCONNECTED**: node monitoring error, possible reasons: heartbeat or guarding failed or an incomprehensible change in the reported state by the node itself. Error handling: use NMT command (*COP_StartNode* or *COP_EnterPreOperational*).

Return Value

Return value	Description
BER_k_ERR	Invalid handle
BER_k_NOT_SENT	Operation not entered in transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Function succeeded
COP_k_NOT_FOUND	No node registered with stated node ID
COP_k_IV	Invalid node ID

Remark

The NMT state of a node is determined via monitoring the communication with the Slave, mainly via heartbeat responses and node guarding responses as well as bootup messages.

9.3 CANopen Object Management

The functions for the CANopen object management are used to create and parameterize CANopen communication objects.

9.3.1 COP_CreatePDO

Creates a PDO in the internal Master Firmware object management.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
node_no	[in]	Node ID of the node allocated to the PDO, valid values 0–127 (0 means not allocated)
pdo_no	[in]	Number of the PDO (1–16). Observe capacity limit values for the CAN interface in use in *Performance Characteristics, p. 106*.
type	[in]	Transmission direction of PDO (from view of Master), possible values: **COP_k_PDO_TYP_RX** for a receive process data object **COP_k_PDO_TYP_TX** for a transmit process data object
mode	[in]	PDO mode according to the coding of subindex 1 (transmission type) of the CANopen communication parameters in the object dictionary, defined values: **COP_k_PDO_MODE_SYNC**: synchronous PDO (corresponds to transmission type 1) **COP_k_PDO_MODE_ASYNC**: asynchronous, event controlled PDO (corresponds to transmission type 254)
length	[in]	Byte length of the PDO (1–8)
CANid	[in]	Identifier of the CAN object that is used by the PDO

Return Value

Return value	Description
BER_k_ERR	Invalid handle
BER_k_NOT_SENT	Operation not entered in transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Function succeeded
COP_k_NOT_FOUND	No node registered with stated node ID
COP_k_CAL_ERR	CAN identifier already in use at receiving side
COP_k_IV	Invalid node ID

Remark

To change the properties of a PDO after the network is started, `COP_CreatePDO` must be called once more for same PDO.

The first four TPDOs and the first four RPDOs according to the Predefined Connection Set are created when calling `COP_AddNode` with data length 8 and transmission type `COP_k_PDO_MODE_ASYNC`.

Depending on the CAN interface in use the number of nodes that can be managed simultaneously varies. USB-to-CAN compact and all 320 based interfaces only support 12+12 PDOs simultaneously. For the exact value see *Performance Characteristics, p. 106*.

9.3.2 COP_DeletePDO

Removes a PDO from the internal Master Firmware object management and releases the resources.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
node_no	[in]	Node ID of the node allocated to the PDO, valid values 0–127 (0 means 0 allocation)
pdo_no	[in]	Number of the PDO (1–16)
type	[in]	Transmission direction of PDO (from view of Master), possible values: **COP_k_PDO_TYP_RX** for a receive process data object **COP_k_PDO_TYP_TX** for a transmit process data object

Return Value

Return value	Description
BER_k_ERR	Invalid handle
BER_k_NOT_SENT	Operation not entered in transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Function succeeded
COP_k_NOT_FOUND	No node registered with stated node ID
COP_k_IV	Invalid node ID

9.3.3 COP_GetPDOInfo

Delivers the PDO properties.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
node_no	[in]	Node ID of the node allocated to the PDO, valid values 0–127 (0 means 0 allocation)
pdo_no	[in]	Number of the PDO (1–16)
type	[in]	Transmission direction of PDO (from view of Master), possible values: **COP_k_PDO_TYP_RX** for a receive process data object **COP_k_PDO_TYP_TX** for a transmit process data object
mode	[out]	PDO mode according to the coding of subindex1 (transmission type) of the CANopen communication parameters in the object dictionary.
length	[out]	Byte length of the PDO (1–8)
CANid	[out]	Identifier of the CAN object that is used by the PDO

Return Value

Return value	Description
BER_k_ERR	Invalid handle
BER_k_NOT_SENT	Operation not entered in transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Function succeeded
COP_k_NOT_FOUND	No node registered with stated node ID
COP_k_IV	Invalid node ID

9.3.4 COP_CreateSDO

If a Server SDO object exists on a registered node, `COP_CreateSDO` creates an additional Client SDO in the internal Master Firmware object management. For each node, that is registered with *`COP_AddNode`* the default SDO exists. An appropriate Server SDO must be provided on the node. This Server SDO has to be configured via the respective object dictionary entry by the Client application.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
node_no	[in]	Node ID of the node allocated to the SDO, valid values 1–127
sdo_no	[in]	Number of the SDO, always **`COP_k_USERDEFINED_SDO`**, because maximum two SDOs per node can be managed and the firmware automatically sets up the default SDO **`COP_k_DEFAULT_SDO`** for each registered node.
clientCANid	[in]	Identifier of the CAN object that is used by the SDO Client for the request to the Server
serverCANid	[in]	Identifier of the CAN object that is used by the SDO Server for the response

Return Value

Return value	Description
`BER_k_ERR`	Invalid handle
`BER_k_NOT_SENT`	Operation not entered in transmit command queue
`BER_k_TIMEOUT`	No response from Master Firmware
`COP_k_OK`	Function succeeded
`COP_k_NOT_FOUND`	No node registered with stated node ID
`COP_k_CAL_ERR`	Server CAN ID not available
`COP_k_IV`	Invalid node ID
`COP_k_SDO_RUNNING`	Ongoing SDO transfer, user defined SDO may not be changed

9.3.5 COP_GetSDOInfo

Delivers the properties of a Client SDO for SDO communication with a registered node.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
node_no	[in]	Node ID of the node allocated to the SDO, valid values 1–127
sdo_no	[in]	Number of the SDO, possible values: **COP_k_DEFAULT_SDO** for the automatically established SDO **COP_k_USERDEFINED_SDO** for the additional user defined SDO
clientCANid	[out]	Identifier of the CAN object that is used by the SDO Client for the request to the Server
serverCANid	[out]	Identifier of the CAN object that is used by the SDO Server for the response

Return Value

Return value	Description
BER_k_ERR	Invalid handle
BER_k_NOT_SENT	Operation not entered in transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Function succeeded
COP_k_NOT_FOUND	No node registered with stated node ID
COP_k_IV	Invalid node ID

9.3.6 COP_SetSDOTimeOut

Defines how long the CANopen Master Firmware waits for the individual response of the SDO Server with a running SDO transfer. Default value is 200 ms.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
w_timeout	[in]	Value of delay time in milliseconds

Return Value

Return value	Description
BER_k_ERR	Invalid handle
BER_k_NOT_SENT	Operation not entered in transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Function succeeded
COP_k_CAL_ERR	General error of Master Firmware

Remark

For more information about the value range and the resolution of the parameter *w_timeout* see *Timer Resolutions and Value Ranges, p. 127*.

9.3.7 COP_DefSyncObj

Defines the cycle time for the system service of the synchronization object.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
sync_period	[in]	Value for the cycle time in milliseconds, valid values 2–65280. The cycle time defines the time between successive synchronization objects.
sync_window	[in]	Reserved, value 0 is invalid, use same vale as in *sync_period*.
CounterOverflow	[in]	Maximum value of the CANopen SYNC counter, valid values 2–240. 0 deactivates the SYNC counter. With activated SYNC counter the SYNC message is transmitted with an additional data byte that is incremented for each transmission. This data byte is reset to 1 every time the maximum value is overstepped. With deactivated SYNC counter the SYNC message is transmitted without the additional data byte.

Return Value

Return value	Description
BER_k_ERR	Invalid handle
BER_k_NOT_SENT	Operation not entered in transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Function succeeded
COP_k_CAL_ERR	General error of Master Firmware
COP_k_IV	Invalid parameter value

Remark

For more information about the value range and the resolution of the parameter *sync_period* see *Timer Resolutions and Value Ranges, p. 127*.

9.3.8 COP_SetSyncDivisor

Sets the divisor for the cycle time for the system service of the synchronization object of the Master Firmware. The parameter *divisor* defines at what internal cycle a SYNC object is transmitted and therefore allows to implement different cycle times for different CAN lines.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
divisor	[in]	Divisor for the intended cycle time for the respective CAN line with the set Master Firmware cycle time that is defined via *COP_DefSyncObj*

Return Value

Return value	Description
BER_k_ERR	Invalid handle
BER_k_NOT_SENT	Operation not entered in transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Function succeeded
COP_k_CAL_ERR	General error of Master Firmware
COP_k_IV	Invalid parameter value

9.3.9 COP_GetSyncInfo

Delivers the properties of the system service of the synchronization object.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
sync_period	[out]	Value for the cycle time in milliseconds. The cycle time defines the time between successive SYNC objects.
sync_window	[out]	Reserved, returns the sync period value
CounterOverflow	[out]	Maximum value of the CANopen SYNC counter, valid values 2–240. 0 deactivates the SYNC counter. With activated SYNC counter the SYNC message is transmitted with an additional data byte that is incremented for each transmission. This data byte is reset to 1 every time the maximum value is overstepped. With deactivated SYNC counter the SYNC message is transmitted without the additional data byte.
divisor	[out]	Divisor for the intended cycle time for the respective CAN line with the configured Master Firmware cycle time that is defined via *COP_DefSyncObj*

Return Value

Return value	Description
BER_k_ERR	Invalid handle
BER_k_NOT_SENT	Operation not entered in transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Function succeeded

Remark

Properties of the system service of the synchronization object can be changed with *COP_DefSyncObj* and *COP_SetSyncDivisor*.

9.3.10 COP_EnableSync

Starts the cyclic transmission of the synchronization object by the Master Firmware.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
mode	[in]	Operating mode of the function, possible values: **COP_k_SINGLE_LINE**: only aimed at the CAN line that is implicitly coded in *boardhdl* **COP_k_ALL_LINES**: aimed at all CAN lines of the interface that is coded in *boardhdl*

Return Value

Return value	Description
BER_k_ERR	Invalid handle
BER_k_NOT_SENT	Operation not entered in transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Function succeeded
COP_k_CAL_ERR	General error of Master Firmware
COP_k_IV	Invalid parameter value

9.3.11 COP_DisableSync

Ends the cyclic transmission of the synchronization object by the Master Firmware.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
mode	[in]	Operating mode of the function, possible values: **COP_k_SINGLE_LINE**: only aimed at the CAN line that is implicitly coded in *boardhdl* **COP_k_ALL_LINEs**: aimed at all CAN lines of the interface that is coded in *boardhdl*

Return Value

Return value	Description
BER_k_ERR	Invalid handle
BER_k_NOT_SENT	Operation not entered in transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Function succeeded
COP_k_CAL_ERR	General error of Master Firmware
COP_k_IV	Invalid parameter value

9.3.12 COP_InitTimeStampObj

Transfers the current time for the system service of the central time information.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
ms	[in]	Time in milliseconds after midnight
days	[in]	Date in days since January 1st, 1984

Return Value

Return value	Description
BER_k_ERR	Invalid handle
BER_k_NOT_SENT	Operation not entered in transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Function succeeded
COP_k_BSY	Timestamp queue is full
COP_k_IV	Invalid parameter value

9.3.13 COP_StartStopTSObj

Starts and stops the cyclic transmission of the central time information (Time Stamp Object) by the Master Firmware. Before calling, make sure that the current time for the system service of the central time information is transferred with *COP_InitTimeStampObj*.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
startstop	[in]	Switch for the Time Stamp Object, possible values: **COP_k_TS_START**: starts transmission **COP_k_TS_STOP**: stops transmission
cycle	[in]	Cycle time in milliseconds, defines the interval of consecutive time information.

Return Value

Return value	Description
BER_k_ERR	Invalid handle
BER_k_NOT_SENT	Operation not entered in transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Function succeeded
COP_k_CAL_ERR	General error of Master Firmware

Remark

Observe information about the cycle time (parameter *cycle*) in *Timer Resolutions and Value Ranges, p. 127*.

9.3.14 COP_GetTimeStampObj

Delivers the properties and the current value of the system service of the central time information (Time Stamp Object).

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
startstop	[out]	Status for the Time Stamp Object, possible values: **COP_k_TS_START**: transmission active **COP_k_TS_STOP**: transmission disabled
cycle	[out]	Cycle time in milliseconds, defines the interval of consecutive time information.
ms	[out]	Time in milliseconds after midnight
days	[out]	Date in days since January 1st, 1984

Return Value

Return value	Description
BER_k_ERR	Invalid handle
BER_k_NOT_SENT	Operation not entered in transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Function succeeded

9.4 CANopen Communication

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

9.4.1 COP_ReadPDO

Reads the data of a process data object (PDO) received by the Master Firmware from the RPDO queue.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
node_no	[out]	Node ID of the CANopen device that transmitted the PDO (valid values 1–127)
pdo_no	[out]	Number of the RPDO, 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 counter of the SYNC object when receiving the PDO, for a description of the SYNC counter see *COP_DefSyncObj*

Return Value

Return value	Description
`BER_k_ERR`	Invalid handle
`COP_k_OK`	Function succeeded
`COP_k_QUEUE_EMPTY`	No new PDOs available
`COP_k_IV`	NULL pointer as parameter

9.4.2 COP_ReadPDO_S

Reads the data of a process data object (PDO) received by the Master Firmware from the RPDO queue and returns the PDO data as a structure.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
sp_pdo	[out]	Pointer to a buffer provided by the Client application of data type `COP_t_RX_PDO`, that accepts the PDO data

COP_t_RX_PDO (Alignment 1 byte)

Field	Type	Description
node_no	`BYTE`	Node ID of the CANopen device that transmitted the PDO (valid values 1–127)
pdo_no	`BYTE`	Number of the RPDO, beginning with 1
length	`BYTE`	Number of valid bytes of `rxdata`
SyncCounter	`BYTE`	Value of the SYNC counter of the SYNC object when receiving the PDO, for a description of the SYNC counter see *COP_DefSyncObj*
a_data[8]	`BYTE[]`	Received PDO data

Return Value

Return value	Description
`BER_k_ERR`	Invalid handle
`COP_k_OK`	Function succeeded
`COP_k_QUEUE_EMPTY`	No new PDOs available
`COP_k_IV`	NULL pointer as parameter

9.4.3 COP_RequestPDO

Initiates a request for a process data object (PDO). The data of the requested PDO are then received via the RPDO queue.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
node_no	[out]	Node of the registered CANopen device that transmits the PDO (valid values 1–127)
pdo_no	[out]	Number of the PDO, beginning with 1

Return Value

Return value	Description
BER_k_ERR	Invalid handle
BER_k_NOT_SENT	Operation not entered in the transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Function succeeded
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 Master Firmware
COP_k_IV	NULL pointer as parameter

9.4.4 COP_WritePDO

Writes the data of a process data object to be transmitted by the Master Firmware into the TPDO queue.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
node_no	[in]	Node ID of the CANopen device to which the PDO is transmitted (valid values 1–127)
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 Value

Return value	Description
BER_k_ERR	Invalid handle
BER_k_NOT_SENT	Operation not entered in the transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Function succeeded
COP_k_BSY	Transmit queue of the Firmware is full.
COP_k_IV	Invalid parameter

Remark

Since the checking of the parameters *node_no* and *pdo_no* is done in the firmware, the function COP_WritePDO does not return an error code if the parameters are incorrect. To check the parameters the event queue has to be read with *COP_GetEvent*. In case of invalid parameters COP_k_WPDO_EVT is returned.

9.4.5 COP_WritePDO_S

Writes the data of a process data object to be transmitted by the Master Firmware into the TPDO queue. Accepts the PDO data as a structure.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
sp_pdo	[in]	Pointer to a buffer provided by the Client application of data type `COP_t_TX_PDO`, that contains the PDO data

COP_t_TX_PDO (Alignment: 1 byte)		
Field	**Type**	**Description**
node_no	`BYTE`	Node ID of the CANopen device to which the PDO is transmitted (valid values 1–127)
pdo_no	`BYTE`	Number of the TPDO, beginning with 1
a_data[8]	`BYTE[]`	PDO data to be transmitted

Return Value

Return value	Description
`BER_k_ERR`	Invalid handle
`BER_k_NOT_SENT`	Operation not entered in the transmit command queue
`COP_k_OK`	Function succeeded
`COP_k_BSY`	Transmit queue of the firmware is full.
`COP_k_IV`	Invalid parameter or NULL pointer

Remark

Since the checking of the parameters *node_no* and *pdo_no* is done in the firmware, the function `COP_WritePDO` does not return an error code if the parameters are incorrect. To check the parameters the event queue has to be read with *`COP_GetEvent`*. In case of invalid parameters `COP_k_WPDO_EVT` is returned.

9.4.6 COP_ReadSDO

Reads the contents of an object dictionary entry from a node. The function works synchronously, therefore the call only returns to the Client application if the SDO transfer is finished. The call is queued in multithreading applications.

```
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 );
```

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
node_no	[in]	Node number to read the object dictionary from (valid values 1–127)
sdo_no	[in]	Number of the SDO to be used. The following values are possible: **COP_k_DEFAULT_SDO**: default SDO for the node, that is automatically created by the firmware is used. **COP_k_USERDEFINED_SDO**: additional SDO, created previously by *COP_CreateSDO* is used.
mode	[in]	Definition of the SDO transmission protocol. With more than 4 bytes of data to be transmitted, the reference data are divided into 7 byte segments. The following values are possible: **COP_k_NO_BLOCKTRANSFER**: (conventional) domain protocol is used, the reception of each segment is confirmed. **COP_k_BLOCKTRANSFER**: block transfer protocol is used, confirmation is given only after max. 127 segments. Implementation of block transfer is optional and not supported by every node.
idx	[in]	16 bit index of the object dictionary entry to be read
subidx	[in]	8 bit index of the object dictionary entry to be read
rxlen	[in/out]	Size of the receive buffer `rxdata`. If the receive buffer is insufficient, neither a separate error code is returned nor is the SDO transport aborted, but the number of required bytes is returned to the Client application in this parameter. If the buffer is full, further received bytes are rejected to prevent an internal buffer overrun.
rxdata	[out]	Address of a sufficient large buffer for the received object dictionary data
abortcode	[out]	Abort code of the SDO transfer (optional parameter)

Return Value

Return value	Description
`BER_k_ERR`	Invalid handle
`Ber_k_MEM_ALLOC_ERR`	Internal data structure or operating system objects not created
`BER_k_SDO_THREAD_ERR`	Error of control of the SDO thread
`BER_k_PC_MC_COMM_ERR`	Communication link with the CAN interface is disturbed
`BER_k_TIMEOUT`	SDO task not taken by the firmware within the communication timeout period (due to SDO engine busy or overload state). Retry later.
`COP_k_OK`	Function succeeded
`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 is included in `abortcode`.
`COP_k_QUEUE_EMPTY`	No SDO response of the Master Firmware
`COP_k_TIMEOUT`	No response from the device, Master Firmware aborts SDO transfer

9.4.7 COP_WriteSDO

Writes data into an object dictionary entry of a node. The function works synchronously, therefore the call only returns to the Client application if the SDO transfer is finished. The call is queued in multithreading applications

```
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 );
```

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
node_no	[in]	Node number to write the object dictionary entry to (valid values 1–127)
sdo_no	[in]	Number of the SDO to be used. The following values are possible: **COP_k_DEFAULT_SDO**: default SDO for the node, that is automatically created by the firmware is used. **COP_k_USERDEFINED_SDO**: additional SDO, created previously by *COP_CreateSDO* is used.
mode	[in]	Definition of the SDO transmission protocol. With more than 4 bytes of data to be transmitted, the reference data are divided into 7 byte segments. The following values are possible: **COP_k_NO_BLOCKTRANSFER**: (conventional) domain protocol is used, the receipt of each segment is confirmed. **COP_k_BLOCKTRANSFER**: block transfer protocol is used, confirmation is given only after max. 127 segments. Implementation of block transfer is optional and not supported by every node.
idx	[in]	16 bit index of the object dictionary entry to be written
subidx	[in]	8 bit index 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 that contains the object dictionary data
abortcode	[out]	Abort code of the SDO transfer (optional parameter)

Return Value

Return value	Description
`BER_k_ERR`	Invalid handle
`Ber_k_MEM_ALLOC_ERR`	Internal data structure or operating system objects not created
`BER_k_SDO_THREAD_ERR`	Error of control of the SDO thread
`BER_k_PC_MC_COMM_ERR`	Communication link with the CAN interface is disturbed
`BER_k_TIMEOUT`	SDO task not taken by the firmware within the communication timeout period (due to SDO engine busy or overload state). Retry later.
`COP_k_OK`	Function succeeded
`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 is included in `abortcode`.
`COP_k_QUEUE_EMPTY`	No SDO response of the Master Firmware
`COP_k_TIMEOUT`	No response from the device, Master Firmware aborts SDO transfer

9.4.8 COP_PutSDO

Initiates reading or writing of a service data object (SDO) by placing an SDO operation in the transmit SDO queue.

COP_PutSDO is a legacy function that is replaced by COP_ParallelPutSDO().

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
node_no	[in]	Node number to read the object dictionary entry from (valid values 1–127)
sdo_no	[in]	Number of the SDO to be used. The following values are possible: **COP_k_DEFAULT_SDO**: default SDO for the node, that is automatically created by the firmware is used. **COP_k_USERDEFINED_SDO**: additional SDO, created previously by *COP_CreateSDO* is used.
mode	[in]	Definition of the SDO transmission protocol. With more than 4 bytes of data to be transmitted, the reference data are divided into 7 byte segments. The following values are possible: **COP_k_NO_BLOCKTRANSFER**: (conventional) domain protocol is used, the receipt of each segment is confirmed. **COP_k_BLOCKTRANSFER**: block transfer protocol is used, confirmation is given only after max. 127 segments. Implementation of block transfer is optional and not supported by every node.
rwAccess	[in]	Transmission direction (read/write). The following values are possible: COP_k_SDO_DOWNLOAD: writing object dictionary access (data are transmitted to the node by the Master Firmware) COP_k_SDO_UPLOAD: reading object dictionary access (data are transmitted from the node to the Master Firmware)
idx	[in]	Index of the object dictionary entry to be written
subidx	[in]	Subindex of the object dictionary entry to be written
length	[in]	For **COP_k_SDO_DOWNLOAD**: size of the buffer data
data	[in]	For **COP_k_SDO_DOWNLOAD**: address of the buffer that contains the data to be transmitted
h_Event	[in]	Optional handle of an operating system event. The Windows event must be created by the Client applications with CreateEvent in the initial state non-signaled.

Return Value

Return value	Description
BER_k_ERR	Invalid handle
BER_k_TIMEOUT	SDO task not taken by the firmware within the communication timeout period (due to SDO engine busy or overload state). Retry later.
COP_k_OK	Function succeeded
COP_k_IV	Invalid parameter value

9.4.9 COP_ParallelPutSDO

Initiates reading or writing of a service data object (SDO) by placing an SDO operation in the transmit SDO queue.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
cookie	[out]	Unique identifier of SDO transfer, required for the corresponding call of `COP_ParallelGetSDO()`
node_no	[in]	Number of the node whose object dictionary is to be accessed (valid values 1–127)
sdo_no	[in]	Number of the SDO to be used. The following values are possible: **`COP_k_DEFAULT_SDO`**: default SDO for the node, that is automatically created by the firmware is used. **`COP_k_USERDEFINED_SDO`**: additional SDO, created previously by *`COP_CreateSDO`* is used.
mode	[in]	Definition of the SDO transmission protocol. With more than 4 bytes of data to be transmitted, the reference data are divided into 7 byte segments. The following values are possible: **`COP_k_NO_BLOCKTRANSFER`**: (conventional) domain protocol is used, the receipt of each segment is confirmed. **`COP_k_BLOCKTRANSFER`**: block transfer protocol is used, confirmation is given only after max. 127 segments. Implementation of block transfer is optional and not supported by every node.
rwAccess	[in]	Transmission direction (read/write). The following values are possible: `COP_k_SDO_DOWNLOAD`: writing object dictionary access (data are transmitted to the node by the Master Firmware) `COP_k_SDO_UPLOAD`: reading object dictionary access (data are transmitted from the node to the Master Firmware)
idx	[in]	Index of the object dictionary entry to be written
subidx	[in]	Subindex of the object dictionary entry to be written
length	[in]	For **`COP_k_SDO_DOWNLOAD`**: size of the buffer `data`
data	[in]	For **`COP_k_SDO_DOWNLOAD`**: address of the buffer that contains the data to be transmitted
h_Event	[in]	Optional handle of an operating system event. The Windows event must be created by the Client applications with `CreateEvent` in the initial state non-signaled. The Linux pthread condition (plus the mutex) must be created by the client application and the mutex must be locked by the calling thread using `pthread_mutex_lock()` before calling `COP_ParallelPutSDO`.

Return Value

Return value	Description
`BER_k_ERR`	Invalid handle
`BER_k_TIMEOUT`	SDO task not taken by the firmware within the communication timeout period (due to SDO engine busy or overload state). Retry later.
`COP_k_OK`	Function succeeded
`COP_k_IV`	Invalid parameter value

Remark

`COP_ParallelPutSDO` is not blocking (asynchronous) and returns immediately to the client application. When the SDO transfer is completed the result must be read with *COP_ParallelGetSDO* The optional handle h_Event can be used to wait synchronously for the end of SDO transfer with `WaitForSingleObject()` or `pthread_cond_timedwait()`.

9.4.10 COP_GetSDO

Reads the result of an SDO transfer previously initialized with *COP_PutSDO* from the receive SDO queue.

COP_GetSDO is a legacy function that is replaced by COP_ParallelGetSDO().

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
length	[in/out]	Size of the receive buffer `data` or number of transmitted bytes
data	[out]	Address of a buffer for the received object dictionary data
abortcode	[out]	Abort code of the SDO transfer (optional parameter)

Return Value

Return value	Description
`BER_k_ERR`	Invalid handle
`BER_k_SDO_THREAD_ERR`	SDO cancelled with *COP_CancelSDO*
`BER_k_DATA_CORRUPT`	Corrupted data received, communication path (USB, Ethernet) disturbed, automatic retry
`BER_k_PC_MC_COMM_ERR`	Communication link with the CAN interface is disturbed
`BER_k_TIMEOUT`	No response from Master Firmware
`COP_k_OK`	Function succeeded
`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 is included in `abortcode`.
`COP_k_SDO_RUNNING`	Currently running SDO transfer, approximate progress in % is shown in `length`
`COP_k_QUEUE_EMPPTY`	No SDO response of Master Firmware available
`COP_k_TIMEOUT`	No response from the device, SDO transfer aborted by Master Firmware

9.4.11 COP_ParallelGetSDO

Reads the result of an SDO transfer previously initialized with *COP_ParallelPutSDO* from the receive SDO queue.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
cookie	[out]	Unique identifier of SDO transfer, returned by *COP_ParallelPutSDO*
length	[in/out]	Size of the receive buffer `data` or number of transmitted bytes
data	[out]	Address of a buffer for the received object dictionary data
abortcode	[out]	Abort code of the SDO transfer (optional parameter)

Return Value

Return value	Description
`BER_k_ERR`	Invalid handle
`BER_k_SDO_THREAD_ERR`	SDO cancelled with *COP_CancelSDO*
`BER_k_DATA_CORRUPT`	Corrupted data received, communication path (USB, Ethernet) disturbed, automatic retry
`BER_k_PC_MC_COMM_ERR`	Communication link with the CAN interface is disturbed
`BER_k_TIMEOUT`	No response from Master Firmware
`COP_k_OK`	Function succeeded
`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 is included in `abortcode`.
`COP_k_SDO_RUNNING`	Currently running SDO transfer, approximate progress in % is shown in `length`
`COP_k_QUEUE_EMPPTY`	No SDO response of Master Firmware available
`COP_k_TIMEOUT`	No response from the device, SDO transfer aborted by Master Firmware

Remark

`COP_ParallelGetSDO` is not blocking (asynchronous) and returns immediately to the client application. The event handle transferred with *COP_ParallelPutSDO* can be used to wait for the end of the SDO transfer..

9.4.12 COP_CancelSDO

Cancels a running SDO operation.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
node_no	[in]	Node ID of the CANopen device whose object dictionary is accessed (valid values 1–127)
sdo_no	[in]	Number of the SDO to be used. The values `COP_k_DEFAULT_SDO` and `COP_k_USERDEFINED_SDO` are possible.
idx	[in]	Index of the object dictionary entry to be written
subidx	[in]	Subindex of the object dictionary entry to be written

Return Value

Return value	Description
`BER_k_ERR`	Invalid handle
`BER_k_NOT_SENT`	Operation not entered in the transmit command queue
`BER_k_TIMEOUT`	No response from Master Firmware
`COP_k_OK`	Function succeeded
`COP_k_IV`	Invalid parameter value
`COP_k_NOT_FOUND`	Nod node registered with the stated node ID

Remark

With the values `node_no`=0 and `sdo_no`=0 all currently running SDO tasks are cancelled simultaneously.

9.4.13 COP_GetEmergencyObj

Reads an emergency object from the EMCY queue and returns the emergency object subdivided into error value, error register and error data.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
node_no	[out]	Node ID of the CANopen device that issued the error message (valid values 1–127)
err_value	[out]	Error code of the error message, 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 Value

Return value	Description
BER_k_ERR	Invalid handle
COP_k_OK	Function succeeded
COP_k_IV	NULL pointer as parameter
COP_k_QUEUE_EMPTY	No new emergency object available

9.4.14 COP_GetEmergencyObj_S

Reads an emergency object from the EMCY queue and returns the alarm message as a structure.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
sp_emergency	[out]	Pointer to a buffer that is provided by the Client application. The buffer is of data type `COP_t_EMERGENCY_OBJ` and receives the alarm message

COP_t_EMERGENCY_OBJ (Alignment: 1 byte)

Field	Type	Description
err_value	`WORD`	Error code of the error message, error codes are standardized according to CiA-301.
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 issued the error message (valid values 1–127)

Return Value

Return value	Description
`BER_k_ERR`	Invalid handle
`COP_k_OK`	Function succeeded
`COP_k_IV`	NULL pointer as parameter
`COP_k_QUEUE_EMPTY`	No new emergency object available

9.4.15 COP_CheckSync

Checks whether a synchronization object is signaled by the Master Firmware.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
SyncCounter	[out]	Value of the SYNC counter of the SYNC object when the SYNC object is transmitted, for a description of the SYNC counter see *COP_DefSyncObj*

Return Value

Return value	Description
BER_k_ERR	Invalid handle
BER_k_PC_MC_COMM_ERR	Communication link with CAN interface disturbed
COP_k_OK	Function succeeded
COP_k_QUEUE_EMPTY	No SYNC object available

9.4.16 COP_GetEvent

Reads a network or a Master Firmware event from the event queue.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
evt_type	[out]	Type of the event, the following values are possible: COP_k_NMT_EVT: network event COP_k_DLL_EVT: local event of the DLL 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 to Flying Master functionality
evt_data1	[out]	Additional information about the event, depending on contents of parameter evt_type
evt_data2	[out]	Additional information about the event, depending on contents of parameter evt_type
evt_data3	[out]	Additional information about the event, depending on contents of parameter evt_type

evt_type = COP_k_NMT_EVT

evt_data1 (cause of the event)	evt_data2	evt_data3
COP_k_NMT_GUARDERR: guard error, no response from device or signaled node status is unexpected.	Node ID of the involved device	Signaled node state: COP_k_NS_STOPPED COP_k_NS_OPERATIONAL COP_k_NS_PREOPERATIONAL COP_k_NS_DISCONNECTED (see also COP_GetNodeState)
COP_k_NMT_BOOTIND: device transmitted a bootup message.		
COP_k_NMT_HEARTBEATERR: heartbeat error, device transmitted nothing or signaled node status is unexpected.		

evt_type = COP_k_DLL_EVT

evt_data1 (status flags of the data link layer, bit coded)	evt_data2	evt_data3
COP_k_DLL_NOERR: error free CAN line, CAN controller is able to communicate	Unused	Unused
COP_k_DLL_COVR: overrun of receive queue of CAN controller		
COP_k_DLL_BOFF: CAN controller in Bus Off		
COP_k_DLL_ESET: error status bit of the CAN controller is set		
COP_k_DLL_ERESET: error status bit of the CAN controller is reset		
COP_k_DLL_RXOVR: overrun of the Firmware internal receive queue		

evt_type = COP_k_DLL_EVT (continued)

evt_data1 (status flags of the data link layer, bit coded)	evt_data2	evt_data3
COP_k_DLL_TXOVR: overrun of the Firmware internal transmit queue		

evt_type = COP_k_WPDO_EVT or COP_k_RPDO_EVT

evt_data1 (event in connection with PDO queues)	evt_data2	evt_data3
COP_k_ERR_PDO_IV: invalid parameter in transmit PDO operation (triggered by COP_WritePDO)	Node ID of the involved device (node_no)	Number of the involved PDO (pdo_no)
COP_k_ERR_PDO_OVR: overrun of the firmware internal transmit or receive queue, corresponding PDO is lost. Additionally COP_k_DLL_EVT with evt_data1=COP_k_DLL_RXOVR and COP_k_DLL_TXOVR are generated.		

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
COP_k_FLY_MASTER: Firmware has network mastership and is the active NMT Master	Unused	Unused
COP_k_FLY_NOT_MASTER: Firmware is not active NMT Master		
COP_k_FLY_LOST_MASTERSHIP: Firmware is replaced by a higher priority NMT Master and loses network mastership		
COP_k_FLY_LOST_ACTIVE_MASTER: active NMT Master failed. Negotiation of the new NMT Master begins with all other potential Masters in the Network.		
COP_k_FLY_WAIT_BUSCONNECTION: Master Firmware is not at CAN		
COP_k_FLY_NEGOTIATION_RUNNING: running negotiation of network mastership with other potential Masters		

Return Value

Return value	Description
BER_k_ERR	Invalid handle
BER_k_PC_MC_COMM_ERR	Communication link with CAN interface disturbed
COP_k_OK	Function succeeded
COP_k_QUEUE_EMPTY	No new entries available
COP_k_IV	NULL pointer as parameter

9.5 LSS Services

The LSS services in accordance with CiA 305 *Layer Settings Services and Protocol (LSS)* are used to configure the parameters of the CANopen network for devices without a direct user interfaces (such as DIP switches). Not all devices support all LSS services.

9.5.1 COP_SetLSSTimeOut

Defines the delay time that determines how long a device response is awaited after transmitting an LSS command. The default value for the delay time is 100 ms.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
w_timeout	[in]	Value for the delay time in milliseconds (valid values 1–32767), values smaller than 5 are rounded up to 5 internally.

Return Value

Return value	Description
BER_k_ERR	Invalid handle
BER_k_NOT_SENT	Operation not entered in the transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Function succeeded
COP_k_IV	Unauthorized parameter value

9.5.2 COP_LSS_InquireAddress

Reads the LSS address of a CANopen device.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
baudtable	[in]	Select the bit timing table to be used for the CAN communication during the LSS communication process. Two tables are possible: COP_k_BAUD_CIA: table with bit timings specified in CiA Standard table. COP_k_BAUD_USER: table with user defined bit timing settings. Depends on hardware if supported, for more information see header file *CopUserBittiming.h*.
baudrate	[in]	Bitrate for the CAN communication during the LSS configuration process, the following values are possible: 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 overwrite the bitrate that is configured with *COP_InitInterface*.
VendorId	[out]	Address of a DWORD buffer that is provided by the Client application. The buffer receives the vendor ID of the device.
ProductCode	[out]	Address of a DWORD buffer that is provided by the Client application. The buffer receives the product code of the device.
RevisionNo	[out]	Address of a DWORD buffer that is provided by the Client application. The buffer receives the revision number of the device.
SerialNo	[out]	Address of a DWORD buffer that is provided by the Client application. The buffer receives the serial number of the device.

Return Value

Return value	Description
BER_k_ERR	Invalid handle
BER_k_NOT_SENT	Operation not entered in the transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Function succeeded
COP_k_NO	General error of the Master Firmware
COP_k_TIMEOUT	No response from the 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

9.5.3 COP_LSS_InquireNodeID

Inquires the node ID of the CANopen device.

```
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 );
```

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
baudtable	[in]	Select the bit timing table to be used for the CAN communication during the LSS communication process. Two tables are possible: `COP_k_BAUD_CIA`: table with bit timings specified in CiA Standard table. `COP_k_BAUD_USER`: table with user defined bit timing settings. Depends on hardware if supported, for more information see header file *CopUserBittiming.h*.

baudrate	[in]	Bitrate for the CAN communication during the LSS configuration process, the following values are possible:
		`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 overwrite the bitrate that is configured with *COP_InitInterface*.

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 (valid values 1–127, special value 255)

Return Value

Return value	Description
`BER_k_ERR`	Invalid handle
`BER_k_NOT_SENT`	Operation not entered in the transmit command queue
`BER_k_TIMEOUT`	No response from Master Firmware
`COP_k_OK`	Function succeeded
`COP_k_NO`	General error of the Master Firmware
`COP_k_TIMEOUT`	No response from the 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

9.5.4 COP_LSS_ConfigNodeID

Sets the node ID of the CANopen device via LSS services.

```
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 );
```

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
baudtable	[in]	Select the bit timing table to be used for the CAN communication during the LSS communication process. There tables are possible: `COP_k_BAUD_CIA`: table with bit timings specified in CiA Standard table. `COP_k_BAUD_USER`: table with user defined bit timing settings. Depends on hardware if supported, for more information see header file *CopUserBittiming.h*.

baudrate	[in]	Bitrate for the CAN communication during the LSS configuration process, the following values are possible:
		`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 overwrite the bitrate that is configured with *COP_InitInterface*.

mode	[in]	Operating mode of the function, possible values: `LSS_k_SET_MODE_SWITCH_MODE_GLOBAL`: LSS inquiry is initialized with a SwitchModeGlobal command `LSS_k_SET_MODE_STORE_CONFIGURATION`: LSS command to save the new mode is 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]	Node ID to be configured on the addressed device (valid values 1–127, special value 255)

Return Value

Return value	Description
BER_k_ERR	Invalid handle
BER_k_NOT_SENT	Operation not entered in the transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Function succeeded
COP_k_NO	General error of the Master Firmware
COP_k_TIMEOUT	No response from the 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

9.5.5 COP_LSS_ConfigBitTiming

Configures the bitrate of the CANopen device via LSS services.

```
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 );
```

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
baudtable	[in]	Select the bit timing table to be used for the CAN communication during the LSS communication process. Two tables are possible: COP_k_BAUD_CIA: table with bit timings specified in CiA Standard table. COP_k_BAUD_USER: table with user defined bit timing settings. Depends on hardware if supported, for more information see header file *CopUserBittiming.h*.
baudrate	[in]	Bitrate for the CAN communication during the LSS configuration process, the following values are possible: 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 overwrite the bitrate that is configured with *COP_InitInterface*.
mode	[in]	Operating mode of the function, possible values: LSS_k_SET_MODE_SWITCH_MODE_GLOBAL: LSS inquiry is initialized with a SwitchModeGlobal command LSS_k_SET_MODE_ACTIVATE_NEW_BAUDRATE: LSS command to activate the new bitrate is transmitted after the delay time specified in switch_delay is expired. LSS_k_SET_MODE_STORE_CONFIGURATION: LSS command to save the new mode is 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]	Bit timing table of the addressed device. COP_k_BAUD_CIA is the standard setting. Manufacturer specific values above 127 are also possible.
new_baudrate	[in]	Bitrate of the addressed device, the following values are possible: 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 in milliseconds before the new bitrate is activated

Return Value

Return value	Description
BER_k_ERR	Invalid handle
BER_k_NOT_SENT	Operation not entered in the transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Function succeeded
COP_k_NO	General error of the Master Firmware
COP_k_TIMEOUT	No response from the 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

9.5.6 COP_LSS_ActivateBitTiming

Switches the bitrate of all connected CANopen devices simultaneously via the LSS service.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
baudtable	[in]	Select the bit timing table to be used for the CAN communication during the LSS communication process. Two tables are possible: `COP_k_BAUD_CIA`: table with bit timings specified in CiA Standard table. `COP_k_BAUD_USER`: table with user defined bit timing settings. Depends on hardware if supported, for more information see header file *CopUserBittiming.h*.

baudrate	[in]	Bitrate for the CAN communication during the LSS configuration process, the following values are possible:
		`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 overwrite the bitrate that is configured with *COP_InitInterface*.

new_baudtable	[in]	Bit timing table of the Master Firmware. `COP_k_BAUD_CIA` and `COP_k_BAUD_USER` are possible values.

new_baudrate	[in]	Bitrate of the Master Firmware, the following values are possible:
		`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 overwrite the bitrate of each individual node that is configured with *COP_LSS_ConfigBitTiming*, but informs the firmware which bitrate it has to set after switching the network bitrate.

switch_delay	[in]	Delay time in milliseconds before the new bitrate is activated

Return Value

Return value	Description
BER_k_ERR	Invalid handle
BER_k_NOT_SENT	Operation not entered in the transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Function succeeded
COP_k_NO	General error of the Master Firmware
COP_k_TIMEOUT	No response from the 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

9.5.7 COP_LSS_IdentifyRemoteSlaves

Identifies devices in the network, if the vendor ID and the product code are known.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
baudtable	[in]	Select the bit timing table to be used for the CAN communication during the LSS communication process. Two tables are possible: `COP_k_BAUD_CIA`: table with bit timings specified in CiA Standard table. `COP_k_BAUD_USER`: table with user defined bit timing settings. Depends on hardware if supported, for more information see header file *CopUserBittiming.h*.

baudrate	[in]	Bitrate for the CAN communication during the LSS configuration process, the following values are possible:
		`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 overwrite the bitrate that is configured with *COP_InitInterface*.

VendorID	[in]	Vendor ID of the device to be identified
ProductCode	[in]	Product code of the device to be identified
RevisionNoLow	[in]	Lower revision number limit of the device to be identified
RevisionNoHigh	[in]	Upper revision number limit of the device to be identified
SerialNoLow	[in]	Lower serial number limit of the device to be identified
SerialNoHigh	[in]	Upper serial number limit of the device to be identified

Return Value

Return value	Description
`BER_k_ERR`	Invalid handle
`BER_k_NOT_SENT`	Operation not entered in the transmit command queue
`BER_k_TIMEOUT`	No response from Master Firmware
`COP_k_OK`	Function succeeded
`COP_k_NO`	General error of the Master Firmware
`COP_k_TIMEOUT`	No response from the 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

9.5.8 COP_LSS_IdentifyNonConfRemoteSlaves

Identifies whether devices in the LSS Waiting State exist in the network.

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

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
baudtable	[in]	Select the bit timing table to be used for the CAN communication during the LSS communication process. Two tables are possible: `COP_k_BAUD_CIA`: table with bit timings specified in CiA Standard table. `COP_k_BAUD_USER`: table with user defined bit timing settings. Depends on hardware if supported, for more information see header file *CopUserBittiming.h*.

baudrate	[in]	Bitrate for the CAN communication during the LSS configuration process, the following values are possible:
		`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 overwrite the bitrate that is configured with *COP_InitInterface*.

Return Value

Return value	Description
`BER_k_ERR`	Invalid handle
`BER_k_NOT_SENT`	Operation not entered in the transmit command queue
`BER_k_TIMEOUT`	No response from Master Firmware
`COP_k_OK`	Function succeeded
`COP_k_NO`	General error of the Master Firmware
`COP_k_TIMEOUT`	No response from the 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

Remark

The delay time until the device response can be configured with *COP_SetLSSTimeOut*.

9.5.9 COP_LSS_Fastscan

Searches devices that are in LSS Waiting State in the network. The function bitwise narrows the LSS address until one device is identified and returns vendor ID, product code, revision number, and serial number of the identified device. If several devices of the same model are present in the network, another call of the function identifies another device.

```
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 );
```

Parameter

Parameter	Dir.	Description
boardhdl	[in]	Handle of CAN interface/line combination
baudtable	[in]	Select the bit timing table to be used for the CAN communication during the LSS scan process. Two tables are possible: `COP_k_BAUD_CIA`: table with bit timings specified in CiA Standard table. `COP_k_BAUD_USER`: table with user defined bit timing settings. Depends on hardware if supported, for more information see header file *CopUserBittiming.h*.

baudrate	[in]	Bitrate for the CAN communication during the LSS scan process, the following values are possible:
		`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 overwrite the bitrate that is configured with *COP_InitInterface*.

VendorID	[in/out]	As input parameter: preset vendor ID to shorten the scan process; or 0 to find any device As output parameter: vendor ID of the detected device
VendorIDBits	[in]	Number of bits to be checked of the given vendor ID starting left at the MSB (value 0–31). For a full range scan without preset vendor ID value in `VendorID`, set the parameter to 31.
ProductCode	[in/out]	As input parameter: preset product code to shorten the scan process; or 0 to find any device As output parameter: product code of the detected device
ProductCodeBits	[in]	Number of bits to be checked of the given product code starting left at the MSB (value 0–31). For a full range scan without preset vendor ID value in `ProductCode`, set the parameter to 31.
RevisionNo	[in/out]	As input parameter: preset revision number to shorten the scan process; or 0 to find any device As output parameter: revision number of the detected device
RevisionNoBits	[in]	Number of bits to be checked of the given revision number starting left at the MSB (value 0–31). For a full range scan without preset vendor ID value in `RevisionNo`, set the parameter to 31.
SerialNo	[in]	As input parameter: preset serial number to shorten the scan process; or 0 to find any device As output parameter: serial number of the detected device
SerialNoBits	[in]	Number of bits to be checked of the given serial number starting left at the MSB (value 0–31). For a full range scan without preset vendor ID value in `SerialNo`, set the parameter to 31.

Return Value

Return value	Description
BER_k_ERR	Invalid handle
BER_k_NOT_SENT	Operation not entered in the transmit command queue
BER_k_TIMEOUT	No response from Master Firmware
COP_k_OK	Function succeeded
COP_k_NO	General error of the Master Firmware
COP_k_TIMEOUT	No response from the 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
LSS_k_FS_NO_ NONCONFIGURED_SLAVE	No response, nothing found
LSS_k_FS_NF_ NONCONFIGURED_SLAVE	No response from device, preset device not found

Remark

The delay time until the device response can be configured with *COP_SetLSSTimeOut*.

Due to the multitude of LSS commands sent on the bus during a fastscan the overall time of the fastcan process is up to 130 times the simple LSS timeout value. For example, with the default LSS timeout of 100 ms a fastscan can take up to 13 seconds. For the amount of time the function does not return to the calling application. Hence the LSS timeout value should be reduced before executing a fastscan.

A Error Codes

A.1 Error Codes of the CANopen Master API DLL

The error codes signal errors in the communication between the Master Firmware and the API DLL as well as internal problems of the API DLL.

Error code name	Error code value	Description	Further information and debugging tips
BER_k_OK	0	Function succeeded	—
BER_k_ERR	1	General error, not further specified, possible reason is an invalid board handle	—
BER_k_EDS_FILENOTFOUND	-44	Device description file not found. *COP_ImportEDS* expects the full path and filename (incl. extension) to an external CANopen device description file. The file is opened in read-only mode.	The given filename must be a UNICODE zero terminated text string.
BER_k_EDS_CORRUPT	-43	Device description file import failed with fatal error. COP_ImportEDS is unable to recognise resp. progress the given file. Severe consistency problems or a truncated device description file are possible reasons.	—
BER_k_EDSLIB	-42	External library *EDSLIB4.dll* is not available.	The import of EDS and DCF files is handled by a dynamically loaded EDS library, that must be present on the system in the same folder as the CANopen Master API library.
BER_k_DATA_CORRUPT	-41	Incorrect data patterns received. This indicates a corruption of the communication path from the firmware to the Windows library (internal error).	Call the function again.
BER_k_NOT_SENT	-40	Congestion of the communication path from the Windows library to the firmware (internal error)	Call the function again after a while.
BER_k_NO_NEW_MSG	-39	Not used	—
BER_k_TIMEOUT	-38	Communication timeout. The firmware has not responded within the communication delay time (CommTimeOut) (internal error).	Check if VCI communication channel is broken.
BER_k_BOARD_ALREADY_USED	-37	Required CAN interface is already used by CANopen Master API. A possible reason is the use of an interpreter language. If aborted during debugging, the interface is not deregistered correctly and the API no longer releases the interface for a further process.	This is a fatal error that stops the Master API operation. Reset the complete DLL with COP_Reset_DLL (only in developing phase).
BER_k_ALL_BOARDS_USED	-36	Master API reached maximum capacity of 12 simultaneously manageable interfaces.	This is a fatal error that stops the Master API operation.
BER_k_BOARD_NOT_SUPP	-35	Requested CAN interface is not supported by CANopen Master API due to unsuitable microcontroller or memory extension.	Use generic mode COP_VCI3GENERIC.
BER_k_BOARD_NOT_FOUND	-34	Specified interface type and identification does not match any CAN interface present on the host system.	This is a fatal error that stops the Master API operation.
BER_k_CANNOT_SEARCH_BOARD	-33	Hardware selection dialog was aborted with **Cancel**.	—
BER_k_WRONG_FW	-32	The version number that is supplied by the firmware is incorrect (internal error). Indicates malfunctioning	This is a fatal error that stops the Master API operation.

Error code name	Error code value	Description	Further information and debugging tips
		communication between PC and microcontroller.	
BER_k_USED_FROM_OTHER_PROCESS	-31	Requested CAN interface is already occupied by another CAN application.	This is a fatal error that stops the Master API operation.
BER_k_PC_MC_COMM_ERR	-30	Communication link with the CAN interface is disturbed.	—
BER_k_BOARD_DLD_ERR	-29	Error during firmware download. Possible reasons: generic VCI firmware library *XatCOP60_VCI3.dll* is missing or can not be loaded; or CAN driver installation problem. This is a fatal error that stops the Master API operation.	Make sure, that the generic VCI firmware library *XatCOP60_VCI3.dll* is located in the same folder as *XatCOP60.dll*. If a CAN driver installation error occurred, observe the installation instructions of the VCI resp. ECI or contact technical support.
BER_k_BADCALLBACK_PTR	-28	Invalid function pointer	—
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 interfaces with an 8 bit microcontroller, because 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. Possible reason: *COP_InitBoard* already called for this CAN line.	The CAN line can be used dynamically. It is possible to release a CAN line that is used in a running program again with *COP_ReleaseBoard* as well as to initialize another one with *COP_InitBoard*.
BER_k_VCI_INST_ERR	-25	Ixxat driver not available, incomplete or defect. This is a fatal error that stops the Master API operation.	Check if interface driver is correctly installed. See section *Installing the Software, p. 17*.
BER_k_BOARD_ERR	-24	Incorrect or unknown interface type.	This is a fatal error that stops the 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 that stops the Master API operation.
BER_k_CCI_INST_ERR	-22	CCI installation error (internal)	Report the hardware version number from the nameplate of the CAN interface to Ixxat support.
BER_k_SDO_INST_ERR	-21	Internal error while instancing or configuring the SDO thread.	This is a fatal error that stops the Master API operation.
BER_k_SDO_THREAD_ERR	-20	Error in the control of the SDO thread or SDO thread cancelled by *COP_CancelSDO*.	Call the function again.

A.2 Error Codes of the CANopen Master Firmware

The following error codes are 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.

Error code name	Error code value	Description
COP_k_OK	0	Function succeeded
COP_k_NO	1	General error, not further specified
COP_k_CAL_ERR	2	General error of the CANopen Master Firmware regarding CAN access
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. Possible reason: node ID not added with *COP_AddNode*
COP_k_NOT_INIT	6	Master not initialized with *COP_InitInterface*
COP_k_INIT	7	Master is initialized and ready to use (valid return code).
COP_k_NO_OBJECTS COP_k_QUEUE_EMPTY	9	No new entries available in data queue (valid return code)
COP_k_TIMEOUT	10	No response from the device
COP_k_CANID_IN_USE	11	Given CAN identifier is already in use.
COP_k_SDO_RUNNING	16	SDO transfer to target node in progress. Wait and retry.
COP_k_BSY	17	CAN transmit queue is full or firmware is busy, for example SDO engine is currently working to capacity. Wait and retry.
COP_k_NO_OBJECT	18	Local object dictionary entry does not exist (internal error), only with SDO manager feature
COP_k_NO_SUBINDEX	19	Local object dictionary entry does not exist (internal error), only with SDO manager feature
COP_k_PRESENT_DEVICE_STATE	21	Access currently not possible, only with SDO manager feature
COP_k_RANGE_EXCEEDED	22	Parameter out of range, 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	Low speed bus coupling not present or not supported for the particular CAN interface

B Performance Characteristics

The following table lists some of the capacity limit values for the supported CAN interfaces. The values are defined by the microcontroller, the memory extension of the CAN interface and the implementation of the CANopen Master Firmware.

	Generic VCI e.g. USB-to-CAN V2	**CAN-IB200 CAN@net II**
Maximum number of supported CAN boards (`COP_InitBoard`)	10	
Maximum number of simultaneously manageable nodes (`COP_AddNode`)	127+1	
Maximum number of TPDOs per node (`COP_CreatePDO`)	16	
Maximum number of RPDOs per node (`COP_CreatePDO`)	16	
Maximum number of SDOs per node (`COP_CreateSDO`)	2 (incl. default SDO)	
Maximum number of parallel running SDO transfers	5	
Maximum total number of RPDOs and TPDOs	1600 (maximum number of PDOs according to Predefined Connection Set defined in CANopen specification/1/ is 1016)	
Maximum number of synchronous RPDOs and TPDOs	600	
Multi line operation (maximum number of supported CAN controllers per board)	Yes (4) SocketCAN: No (concept of virtual standalone network controller)	
Flying Master additional functionality	Yes	

C Scope of Delivery

By default the CANopen Master API is installed in the directory \Programs\Ixxat\CANopen Master API. Sample applications are installed in \Users\Public\Documents\Ixxat CANopen Master API. Neither the program library nor the demo applications enter program settings in the Windows registry.

C.1 Windows

Folder \ (Documentation)	
Files	**Description**
files.txt	List of installed files with version number
LiesMich.txt	Important product information (German)
ReadMe.txt	Important product information (English)
HISTORY.txt	Development history and record of changes
4.12.0132.10000.pdf	Software Design Guide (German)
4.12.0132.20000.pdf	Software Design Guide (English)

Folder \bin (Binary Files)	
Files	**Description**
XatCOP60.dll	CANopen Master API 6
XatCOP60-64.dll	CANopen Master API 6 (64 bit version for processor architecture amd64)
XatCOP_VCI3.dll	Generic VCI3 firmware
XatCOP_VCI3-64.dll	Generic VCI3 firmware (64 bit version for processor architecture amd64)
EDSLIB4.dll	EDS/DCF file parser
EDSLIB4-64.dll	EDS/DCF file parser (64 bit version for processor architecture amd64)
MdlWrapper6.dll	COM wrapper of the CANopen Master API 6.1 for legacy applications

Folder \bin\debuglog (Debug Version)	
Files	**Description**
XatCOP60.dll	CANopen Master API
XatCOP60-64.dll	CANopen Master API (64 bit version for processor architecture amd64)

Folder \lib (Library Files for C Compilers), \li\BCB (Library Files for Borland C++ Builder)	
Files	**Description**
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

Folder \lib\MSVC (Lib files for Microsoft Visual C++ / Visual Studio)	
Files	**Description**
XatCOP60.lib	Lib files for MS Visual C / Visual Studio
XatCOP60-64.lib	Lib files for MS Visual C / Visual Studio (64 bit version)

Folder \Samples	
(Headers for all programming languages with constants, types and interface descriptions)	
Files	**Description**
cop.bas	Visual Basic (legacy) main header of the CANopen Master API

Folder \Samples (continued)

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 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 LSS constants
CopError.h	C header for issuing description texts for return error codes
CopError.c	Corresponding C implementation file
CatBrds.bas	All Ixxat interface types (CAN interfaces) for Visual Basic (legacy)
CatBrds.h	C header of all Ixxat interface types (CAN interfaces VCI 2)
CatBrds.pas	Delphi (Pascal) header off all Ixxat interface types (CAN interfaces VCI 2)
CatBrds.cs	All Ixxat interface types (CAN interfaces) for C#
CatBrds.vb	All Ixxat interface types (CAN interfaces) for Visual Basic.NET
Vciguid.h	C header of all Ixxat interface types (CAN interfaces VCI)
Vci3Guid.pas	Delphi (Pascal) header off all Ixxat interface types (CAN interfaces VCI)

Folder \Samples\C (C Sample Application), \Samples\C\BCB DigIO

(Borland C++ Builder sample application for control of external DS-401 device via two default PDOs of length 1)

Files	**Description**
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 utilized for support of XP visual styles
XatCOP60.dll	CANopen Master API
XatCOP_VCI3.dll	Generic VCI3 firmware
EDSLIB4.dll	EDS/DCF file parser
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

Folder \Samples\C\BCB NetManage

(Borland C++ Builder sample application for object dictionary access to external devices via the default SDO)

Files	**Description**
FileOpen.bmp	Bitmap for button
FileSave.bmp	Bitmap for button
Main.dfm	Configuration of control elements in the main window of the application
Maind.cpp	Implementation of the main window of the application
Main.h	Corresponding header of the main window
NetManage.bpr	BCB project file

Folder \Samples\C\BCB NetManage (continued)

NetManage.cpp	Implementation of WinMain()
NetManage.exe	Release version of the sample application
NetManage.exe.manifest	Declaration of the common controls utilized for support of XP visual styles
XatCOP60.dll	CANopen Master API
XatCOP60_VCI3.dll	Generic VCI3 firmware
EDSLIB4.dll	EDS/DCF file parser
NetManage.res	Binary resource file, contains the application icon
NetManage.ico	Icon of the application

Folder \Samples\C\MFC\DigIO

(MFC sample application for control of external DS-401 device via two default PDOs of length 1 for Visual Studio 2005 and 6)

Files	Description
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 utilized for support of XP visual styles
XatCOP60.dll	CANopen Master API
XatCOP_VCI3.dll	Generic VCI3 firmware
EDSLIB4.dll	EDS/DCF file parser
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 in use
StdAfx.h	Header for configuration of the MFC elements in use

Folder \Samples\C\MFC\DigIO\res (Resource Files)

Files	Description
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

Folder \Samples\C\MFC\DigIO\x64

(64 bit version of the sample application for processor architecture amd64)

Files	Description
DigIOdemo.exe	Release build of the sample application

Folder \Samples\C\MFC\DigIO\x64 (continued)

DigIOdemo.exe.manifest	Declaration of the common controls utilized for support of XP visual styles
XatCOP60-64.dll	CANopen Master API (64 bit version)
XatCOP_VCI3-64.dll	Generic VCI3 firmware (64 bit version)
EDSLIB4-64.dll	EDS/DCF file parser (64 bit version)

Folder \Samples\C\Win32 App

(Sample application for monitoring of an external DS-401 device and control via the default RPDO1 of length 2 developed without class library with the pure Windows API for Visual Studio 2005 and 6)

Files	Description
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 control version used for XP visual styles
XatCOP60.dll	CANopen Master API
XatCOP_VCI3.dll	Generic VCI3 firmware
EDSLIB4.dll	EDS/DCF file parser
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

Folder \Samples\C\Win32 App\x64

(64 bit version of the sample application for processor architecture amd64)

Files	Description
COPMSMPL.exe	Release build of the sample application
COPMSMPL.exe.manifest	Declaration of the common control version used for XP visual styles
XatCOP60-64.dll	CANopen Master API (64 bit version)
XatCOP_VCI3-64.dll	Generic VCI3 firmware (64 bit version)
EDSLIB4-64.dll	EDS/DCF file parser (64 bit version)

Folder \Samples\C\VS2010 MFC DigIO

(MFC sample application for control of external DS-401 device via two default PDOs of length 1 for visual Studio 2010)

Files	Description
DigIOdemo.cpp	Implementation of the application class
DigIOdemo.exe	Release build of the sample application
DigIOdemo.exe.manifest	Declaration of the common controls utilized for support of XP visual styles
XatCOP60.dll	CANopen Master API
XatCOP_VCI3.dll	Generic VCI3 firmware
EDSLIB4.dll	EDS/DCF file parser
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

Folder \Samples\C\VS2010 MFC DigIO (continued)

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 in use
StdAfx.h	Header for configuration of the MFC elements in use

Folder \Samples\C\VS2010 MFC DigIO\res (Resource Files)

Files	Description
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

Folder \Samples\C\VS2010 MFCDigIO\x64

(64 bit version of the sample application for processor architecture amd64)

Files	Description
DigIOdemo.exe	Release build of the sample application
DigIOdemo.exe.manifest	Declaration of the common controls utilized for support of XP visual styles
XatCOP60-64.dll	CANopen Master API (64 bit version)
XatCOP_VCI3-64.dll	Generic VCI3 firmware (64 bit version)
EDSLIB4-64.dll	EDS/DCF file parser (64 bit version)

Folder \Samples\C\VS2010 Win32 App

(Sample application for monitoring of an external DS-401 device and control via the default RPDO1 of length 2 developed without class library with the pure Windows API for Visual Studio 2010)

Files	Description
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 control version used for XP visual styles
XatCOP60.dll	CANopen Master API
XatCOP_VCI3.dll	Generic VCI3 firmware
EDSLIB4.dll	EDS/DCF file parser
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

Folder \Samples\C\VS2010 Win32 App\x64

(64 bit version of the sample application for processor architecture amd64)

Files	Description
COPMSMPL.exe	Release build of the sample application
COPMSMPL.exe.manifest	Declaration of the common control version used for XP visual styles

Folder \Samples\C\VS2010 Win32 App\x64 (continued)

XatCOP60-64.dll	CANopen Master API (64 bit version)
XatCOP_VCI3-64.dll	Generic VCI3 firmware (64 bit version)
EDSLIB4-64.dll	EDS/DCF file parser (64 bit version)

Folder \Samples\C#, \Samples\C#\DigIO

(C# sample application and C# sample application to control an external DS-401 device via two default PDOs of length 1 for Visual Studio 2008 and .NET 2.0 Framework)

Files	Description
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
XatCOP_VCI3.dll	Generic VCI3 firmware
EDSLIB4.dll	EDS/DCF file parser
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 colored status LED
XatBrds.cs	Copy of the Ixxat interface type file (CAN interfaces) for C#

\Samples\C#\DigIO\Properties (Resource Files)

Files	Description
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)

Files	Description
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
XatCOP_VCI3.dll	Generic VCI3 firmware
EDSLIB4.dll	EDS/DCF file parser
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 interface type file (CAN interfaces) for C#

\Samples\C#\NetManage\Properties (Resource Files)

Files	Description
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

Folder \Samples\Delphi, \Samples\Delphi\DigIO

(Delphi sample applications, Delphi sample application for control of external DS-401 device via two default PDOs of length 1)

Files	Description
DigIOdemo.dof	Configuration file of the application
DigIOdemo.dpr	Delphi project file
DigIOdemo.exe	Debug build of the sample application
XatCOP60.dll	CANopen Master API
XatCOP_VCI3.dll	Generic VCI3 firmware
EDSLIB4.dll	EDS/DCF file parser
DigIOdemo.ico	Icon of the application
DigIOdemo.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

Folder \Samples\Delphi\NetManage

(Delphi sample application for object dictionary access to external devices via the default SDO)

Files	Description
FileOpen.bmp	Bitmap for button
FileSave.bmp	Bitmap for button
Main.dfm	Configuration of control elements in the main window of the application
Maind.pas	Implementation of the main window of the application
NetManage.dof	Configuration file of the application

Folder \Samples\Delphi\NetManage (continued)

NetManage.dpr	Delphi project file
NetManage.exe	Debug build of the sample application
XatCOP60.dll	CANopen Master API
XatCOP60_VCI3.dll	Generic VCI3 firmware
EDSLIB4.dll	EDS/DCF file parser
NetManage.res	Binary resource file, contains the application icon
NetManage.ico	Icon of the application

Folder \Samples\Delphi.net

(Headers for Delphi.net with constants, types and interface descriptions)

Files	Description
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

Folder \Samples\VB .NET, \Samples\VB .NET\DigIO

(Visual Basic.NET sample application and Visual Basic.NET sample application to control an external DS-401 device via two default PDOs of length 1 for Visual Studio 2008 and .NET 2.0 Framework)

Files	Description
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
XatCOP_VCI3.dll	Generic VCI3 firmware
EDSLIB4.dll	EDS/DCF file parser
DigIOdemo.ico	Icon of the application
DigIOdemo.sln	Visual Studio 2008 project file
DigIOdemo.vbproj.user	Visual C# 2008 project settings
MainForm.vb	Implementation of the main window of the application
MainForm.resx	Configuration of the 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 interface type file (CAN interfaces) for Visual Basic. NET

\Samples\VB .NET\DigIO\Properties (Resource Files

Files	Description
app.manifest	User account 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

Folder \Samples\LabView (LabView Examples)	
Files	**Description**
IXXAT CANio500.lvproj	LabView example project for controlling of Ixxat CANopen I/O module CANio 500
Readme.txt	Note for using the LabView CLN library

Folder \Samples\LabView\CLN_IXXAT_Master_API_Functions	
(Library of virtual instruments (.vi) using Call Library Function nodes for the single Master API functions	
Files	**Description**
COP_Custom_Error_Codes.vi	Translates the Master API error codes to a LabView compliant range using the 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_ImportEDS.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	

Folder \Samples\LabView\CLN_IXXAT_Master_API_Functions (continued)

COP_StartNode.vi	
COP_StartStopTSObj.vi	
COP_StopNode.vi	
COP_WritePDO.vi	
COP_WritePDO_s.vi	
COP_WriteSDO.vi	
IXXAT_Master_API_PathAndName.vi	Global variable encapsulating the local path and filename to CANopen Master API library depending on bitness etc.
IXXAT_Master_API.mnu	All Master API functions as LabView palette
XatCOP60.dll	CANopen Master API
XatCOP_VCI3.dll	Generic VCI3 firmware
EDSLIB4.dll	EDS/DCF file parser
XatCOP60-64.dll	CANopen Master API (64 bit version)
XatCOP_VCI3-64.dll	Generic VCI3 firmware (64 bit version)
EDSLIB4-64.dll	EDS/DCF file parser (64 bit version)

Folder \Samples\LabView\CLN_IXXAT_Master_API_Functions\Additional Features

(Library of virtual instruments (.vi) using Call Library Function nodes for the single Master API functions of Flying Master additional features)

Files	Description
COP_ConfigFlyMaster.vi	
COP_GetStatusFlyMasterNeg.vi	
COP_StartFlyMaster.vi	

Folder \Samples\LabView\CLN_IXXAT_Master_API_Functions\LSS

(Library of virtual instruments (.vi) using Call Library Function nodes for the single Master API functions of LSS services)

Files	Description
COP_LSS_ActivateBitTiming.vi	
COP_LSS_ConfigBitTiming.vi	
COP_LSS_ConfigNodeID.vi	
COP_LSS_Fastscan.vi	
COP_LSS_IdentifyNonConfRemoteSlaves.vi	
COP_LSS_IdentifyRemoteSlaves.vi	
COP_LSS_InquireAddress.vi	
COP_LSS_InquireNodeID.vi	
COP_LSS_SetLSSTimeOut.vi	

Folder \Samples\LabView\CLN_IXXAT_Master_API_Functions\special

(Library of virtual instruments (.vi) using Call Library Function nodes for the single Master API functions for experts)

Files	Description
COP_PutSDO.vi	
COP_GetSDO.vi	
COP_CancelSDO.vi	

Folder \Samples\LabView\custom

(Manufacturer specific additions to the LabView environment)

Files	Description
IXXAT_Master_API-errors.txt	Master API error codes with LabView compliant offset 6100

Folder \Samples\LabView\examples (LabView examples)	
Files	**Description**
example_find_all_nodes.vi	
IXXAT CANio500.vi	
pdo_statt_lss.vi	
read_sdo_predefined.vi	
write_sdo_predefined.vi	

Folder \Samples\LabView\complex_functions_based_on_API	
(Sub-vis used by the LabView samples and the Master API CLN library)	
Files	**Description**
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	

Folder \Tools (Utilities)	
Files	**Description**
XCFlash.exe	VCI2 flash programmer for Ixxat CAN interfaces
ucii161f.H86	VCI2 flash firmware for use with iPCI-XC16/PCI CAN interface
iPC-I_XC16_PCI_FLASH.H86	VCI3 flash firmware for use with iPCI-XC16/PCI CAN interface
COP_i161xc_2ch_flash.H86	VCI2 CANopen Master API dual channel flash firmware for use with iPCI-XC16 CAN interface
COP_i161xc_2ch_flash_vci3.H86	VCI3 CANopen Master API dual channel flash firmware for use with iPCI-XC16 CAN interface

C.2 Linux

The CANopen Master API is available as zipped archive. If unpacked into base directory */home/user/CANopenMasterAPI* the directory structure is as follows.

./	
Files	**Description**
<xxxxx>	Documentation
Makefile	Installation script, copies the static library files to */usr/lib*

./bin/x64	
(Binary files for x86–64 platform)	
Files	**Description**
libXatCOP60.so	CANopen Master API 6
libXatCOP_sock.so	Generic SocketCAN Firmware (dynamically linked by libXatCOP60.so)

./bin/x64/debuglog	
(Debug version, generates the protocol file CatCOP60.log)	
Files	**Description**
libXatCOP60.so	CANopen Master API 6

./License	
(Software license agreements)	
Files	**Description**
IX_License_HW&SW.txt	Ixxat software license agreement

./License/thirdpaty
Software license agreements of external components used by the product

./Samples	
(Headers for all programming languages with constants, types and interface descriptions)	
Files	**Description**
Cop.h	C main header of the CANopen Master API
Cop.cs	C# header of the CANopen Master API
copcmd.h	C header with the command opcodes of the CANopen Master API
CopSDOManager.h	C header containing additional CANopen Master API functions
LSScmd.h	C header with LSS constants
CopError.h	C header for issuing description texts for the return error codes
CopError.c	Corresponding C implementation file
XatBrds.cs	All Ixxat board types for C#
guiddef.h	C/C++ definition of GUID data type
vciguid.h	C header of all Ixxat board types

./Samples/C#	
(C# sample application for .net core 2.0 platform independent, for use with Visual Studio Code)	
Files	**Description**
Cop.cs	Copy of the CANopen Master API C# main header
Program.cs	C# main program, console application to control a DS-401 device via its two default PDOs of length 1 for dotnet commandline executable and .net core 2.0 framework
ConsoleApp.cs.csproj	.net core project file
XatBrds.cs	Copy of Ixxat board type file for C#

./Samples/C#/Properties	
(Resource files)	
Files	**Description**
launchSettings.json	Specific environment and platform settings used to launch the application

./Samples/C/.vscode	
(Visual Studio Code project settings)	
Files	**Description**
launch.json	Specific platform settings used to launch and debug the application
tasks.json	Build the application via dotnet commandline

./Samples/C	
(C sample application prepared for Visual Studio Code, platform independent)	
Files	**Description**
TestCatCop6.cpp	C main program as console application to test and execute several CANopen Master API functions
cXatCOP6diag.hpp	Static helper class, prints and logs all CANopen Master API function calls including their parameters

./Samples/C/.vscode	
(Visual Studio Code project settings)	
Files	**Description**
c_cpp_properties.json	Development host specific environment and platform settings for C/C++ workload
launch.json	Specific platform settings used to launch and debug the application
tasks.json	Compile and build settings and commands for C++

D Details of Linux Version

The Windows and the Linux version are implemented as shared libraries and share the same sources. Therefore, the range of functions and their behavior is equal for both platforms. Due to operating system dependency of some API functions and missing CAN driver preconditions the Linux version has minor limitations.

The following functions are not available on Linux:

- `COP_DefineMsgRPDO`
- `COP_DefineMsgEvent`
- `COP_DefineMsgEmergency`
- `COP_DefineMsgSync()`

The following function is limited on Linux:

- `COP_InitBoard()`: in parameter `pBoardtype` the value `COP_BOARDDIALOG` is not supported

Precision of the Timer Thread

The precision of the timer thread that is used as base for all CANopen Master API internal timing intervals and build around `clock_gettime( CLOCK_MONOTONIC, struct timespec * )`, is not as accurate as its Windows Multimedia timer counterpart. Even though a timer correction algorithm detects and compensates the timer drift, a jitter is noticeable, and outliers of about ±10 % to the 1 ms timer tick can occur. The regular user is not permitted to change the priority of the timer thread for improved precision. With administrator rights the `nice` value of the application process (https://askubuntu.com/questions/656771/process-niceness-vs-priority) can be adjusted. But that does not really improve real time behavior, as it still limits the highest possible priority to 100, while the real-time priority area starts at 99. If this is not acceptable, the client application might periodically call `COP_ClockTick1ms()` from a real time or high precision (hardware) timer process or thread.

The so-called Linux RT-Preempt patch should also improve application response times. It converts the old Linux timer API into separate infrastructures for high resolution kernel timers plus one for timeouts. This leads to user space POSIX timers with high resolution, and makes in-kernel locking-primitives preemptable. See https://wiki.linuxfoundation.org/realtime/start for more information.

SocketCAN

To set the bit rate, start and stop the CAN controller, reset and restart the CAN controller, etc is only allowed with administrator rights. Therefore most of the CANopen Master API LSS functions do not work with SocketCAN. Function `COP_InitInterface()` returns `COP_k_NOT_INIT`, which means *Error during initialization of CAN controller*, due to insufficient user rights.

If the CANopen user application cannot be run as Administrator, then the CAN controller needs to be appropriately configured prior to starting the application. The CAN bit rate (here: 250 kBit/sec) must be selected and the CAN controller must be set 'up' via commandline:

```
sudo ip link set can0 type can bitrate 250000
sudo ip link set can0
```

Or combined:

```
sudo ip link set can0 up type can bitrate 250000
```

With SocketCAN, there is no grouping into CAN board and CAN line. Each CAN line is operating independently. Therefore the parameter of `COP_InitBoard()` that addresses the CAN line is not necessary and must always be set to `COP_FIRSTLINE`.

To correlate a SocketCAN network interface to a Ixxat CAN board, use one of the following predefined identifiers for the `COP_InitBoard()` call:

- `GUID_SOCKETCAN_CLASS` and `COP_1stBOARD` / `COP_2ndBOARD`
- `GUID_SOCKETCAN_CLASS` and `GUID_SOCKETcan0` / `GUID_SOCKETcan1`
- `COP_k_DEFAULTBOARD`

E Implementation Specific Aspects

E.1 Processing of Synchronous PDOs

TPDOs

PDO data for synchronous PDOs transferred 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. According to the coding of subindex2 of the CANopen PDO communication parameters in the object dictionary the transmission type is configured 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. When the SYNC object is transmitted all PDOs received in the meantime are transferred via the RPDO queue and can be read with `COP_ReadPDO()`.

E.2 Node Guarding and Node States

If a device signals an unexpected node state during node guarding, the Master Firmware generates a network event of type `COP_k_NMT_EVT`. Function `COP_GetEvent()` returns the event cause `COP_k_NMT_GUARDERR` or `COP_k_NMT_HEARTBEATERR` in the parameter *evt_data1* and the node ID in parameter *evt_data2*. If the node state that is signaled by the slave is different to the one set by the client application with the NMT functions and if a slave is not in pre-operational state after its bootup, then an unexpected node state is signalled. The same network events are generated when the involved node is lost, i.e. no longer reacts to the guarding telegram or no longer transmits heartbeat messages. For a better distinction of the various error scenarios, the 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. 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.

E.3 Enhanced Bus Off Detection and Auto Recovery

In the CANopen Master API each physical restriction of CAN communication results in a `COP_k_DLL_BOFF` `COP_k_DLL_EVT` event. The event is reported to the client application for information and then the controller automatically resets and restarts and the communication should start again.

If the CAN line is broken and disconnected the recovery fails and `COP_k_DLL_BOFF` is reported again. The typical interval for bus off detection is 500 ms at 250 kBit/s depending on the bit rate and proportionally prolonged for lower bit rates.

A condition for a bus off detection is an actual CAN transmission request of the CANopen Master API. A bus off can only be detected if the CANopen Master is sending anything (heartbeat message, NMT command, SDO request, PDO or SYNC object).

If the CAN initialization is successful the firmware signals a `COP_k_DLL_NOERR` `COP_k_DLL_EVT` event which holds the currently active bit rate in the data bytes of the event notification once the first CAN message is successfully transmitted by the CAN controller. The event signals that the CAN line is fully functional and can be used by the user application as bus recovery success indicator after a `COP_k_DLL_BOFF` event.

F Communicating with the CANopen Master Firmware

HMS recommends not to implement direct calls to `COP_SendMsg` and `COP_GetMsg` because these API functions will be removed with the next revision of the CANopen Master API.

Almost all functions of the CANopen Master API work internally according to a fixed structure with *`COP_SendMsg`* and *`COP_GetMsg`*.

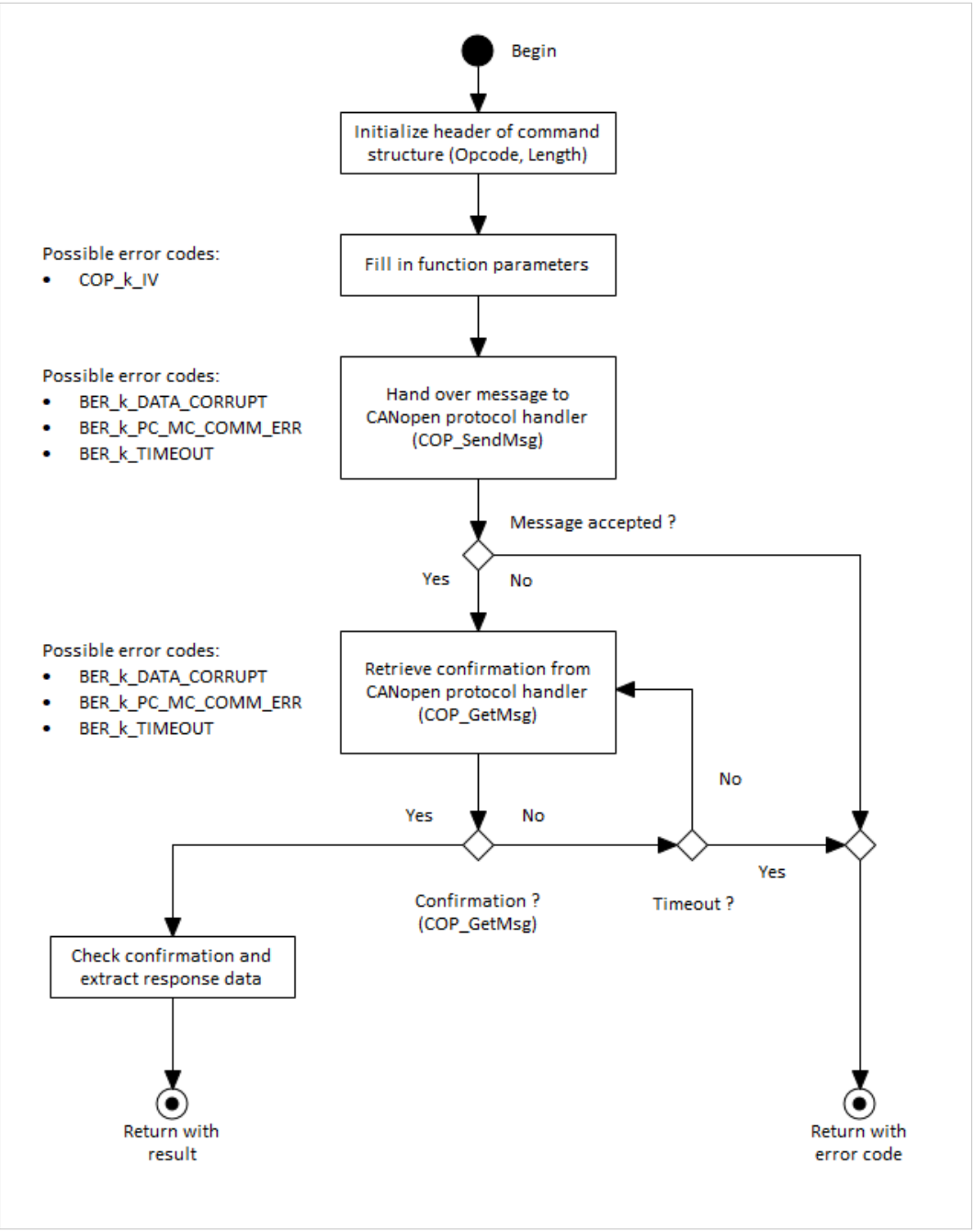

Fig. 3 Internal structure

- The command record `COP_t_Message` is initialized according to the DLL function that is called by the application program.
- Command opcodes and function parameters are entered into `COP_t_Message`.

- The command record `COP_t_Message` is entered in the transmit command queue with `COP_SendMsg`.
- By querying the receive command queue cyclically with `COP_GetMsg` (time restricted) the acknowledgement (confirmation) of the Master Firmware is awaited.
- With *`COP_SetCommTimeOut`* the waiting time for the acknowledgement of the Master Firmware is set (default: 5 s).
- The error code is extracted from the received acknowledgement command and returned to the application program.

G Troubleshooting — Frequent Source of Errors

G.1 Presetting and Initializing the CAN Board

For the unique identification of the desired CAN board the arguments *boardtype* and *boardID* with `COP_InitBoard()` are strictly checked by the Master API DLL and thus must be initialized properly before the call. The delivered unique board identification of the local CAN board is typically the serial number

Example

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

short res = COP_InitBoard( &wBoardhdl, &boardtype, &boardID, 0 );
if( BER_k_OK == res )
{
    char sz[32] = {0};
    sprintf_s( sz, sizeof(sz), "%s", &boardID ); // e.g. HW800511
}
```

If, for example, the boardID is missing, the compiler might set the memory of the variable to zero or the memory might contain random data. In both cases the value is an invalid board identification and the function call results in `BER_k_BOARD_NOT_FOUND`.

G.2 Reading Receive Data Queues

The four receive data queues RPDO queue, EMCY queue, event queue, and SYNC queue must be read regularly with the corresponding function until the queue is empty, because their capacity is relatively small. Reading is possible either synchronously by polling (see *Polling, p. 19*) or asynchronously by callbacks (see *Callbacks, p. 19*).

Asynchronous

If the functions `COP_DefineCallbacks()`, `COP_DefineMsgRPDO`, `COP_DefineMsgEvent`, `COP_DefineMsgEmergency`, or `COP_DefineMsgSync` are called for initialization the firmware can inform when a receive object is entered in the corresponding data queue.

If callback functions or Windows messages are defined, the receive objects must be read from the corresponding queue with the functions `COP_ReadPDO()`, `COP_GetEmergencyObj()`, `COP_GetEvent()`, and `COP_Checksync()`.

Example: How to Read RPDO Queue

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

For the EMCY queue, the event queue and the SYNC queue use the corresponding functions *`COP_GetEmergencyObj`*, *`COP_GetEvent`*, *`COP_CheckSync`* to check each queue.

H Timer Resolutions and Value Ranges

	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 initialization: `COP_InitInterface()` Heartbeat time: `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

Box 4126
300 04 Halmstad, Sweden

info@hms.se
4.12.0132.20000 1.0 en-US / 2019-04-30 / 12996