BAUMÜLLER

be in motion be in motion

BM4-O-CAN-03

CANopen-Slave for b maXX PLC
Application Manual

E	5.03057.02

Title	Application Manual
Product	CANopen-Slave for b maXX PLC BM4-O-CAN-03
Last Revision:	July 19, 2005
Order number	376487

Manufacturer

Baumüller Nürnberg GmbH
Ostendstr. 80 - 90
90482 Nuremberg
Germany
Tel. +49 9 11 54 32 - 0 Fax: +49 9 11 54 32 - 1 30
www.baumueller.de

1

INTRODUCTION

This Application Manual is an important component of your b maXX 4400 device; Hence not least in the interest of your own safety, read through this documentation completely.

In this chapter, we will describe the first steps.

1.1 First Steps

1 To program the CANopen slave option module, you need the following hardware:
a b maXX 4400 basic unit,
a b maXX PLC option module and
CANopen Slave option module.
You must have installed the hardware in accordance with the Operating Instructions in each case and it must be ready for operation.
In order to create a functional CANopen network, you will also need a CANopen Master, communication cable and terminating resistor connector, and possibly additional CANopen Slaves (e.g. additional b maXX 4400 with CANopen Slave option module or I/O modules).

2 Apart from this, you need the following software:
PROPROG wt II for programming the b maXX PLC and the CANopen Slave option module for b maXX PLC and
b maXX Drive-Configurator for PROPROG wt II Version 1.0 (or higher).

1.2 Terms Used

For the Baumüller product "BM4-O-CAN-03" (CANopen Slave option module for b maXX PLC), we will also use the term "CANopen Slave plug-in module" in this documentation. The terms CANopen Slave and network node will also be used for the CANopen Slave option module for b maXX PLC.

For the Baumüller products "BM4-O-ETH-02" (Ethernet with CANopen Master option module for b maXX PLC) and "BM4-O-CAN-04" (CANopen Master option module for b maXX PLC), we will also use the terms "CANopen Master plug-in module" for BM4-O-ETH-02 or BM4-O-CAN-04 in this documentation.

We will also use the term "b maXX" for the "Basic Unit b maXX 4400" product.

The controller in the basic unit is also referred to as the "b maXX controller".

For a list of the abbreviations that are used, refer to ▷Appendix A - Abbreviations.

1.3 Conditions

This manual is based on the "b maXX PLC Application Manual" and assumes that you have knowledge of the PROPROG wt II programming tools and have read its manual.

2

Basic Safety Instructions

We have designed and manufactured each Baumüller plug-in module in accordance with the strictest safety regulations. Despite this, working with the plug-in module can be dangerous for you.

In this chapter, we will describe the risks that can occur when working with a Baumüller plug-in module. Risks are illustrated by icons. All the symbols that are used in this documentation are listed and explained.

In this chapter, we cannot explain how you can protect yourself from specific risks in individual cases. This chapter contains only general protective measures. We will go into concrete protective measures in subsequent chapters directly after information about the individual risk.

2.1 Hazard information and instructions

WARNING

The following **may occur**, if you do not observe this warning information:

- serious personal injury
- death

The hazard information is showing you the hazards which can lead to injury or even to death.

Always observe the hazard information given in this documentation.

Hazards are always divided into three danger classifications. Each danger classification is identified by one of the following words:

DANGER

- Considerable damage to property
- Serious personal injury
- Death **will** occur

WARNING

- Considerable damage to property
- Serious personal injury
- Death **can** occur

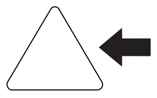
CAUTION

- Damage to property
- Slight to medium personal injury **can** occur

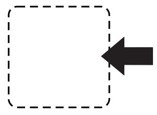
2.1 Hazard information and instructions

2.1.1 Structure of hazard information

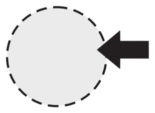
The following two examples show how hazard information is structured in principle. A triangle is used to warn you about danger to living things. If there is no triangle, the hazard information refers exclusively to damage to property.



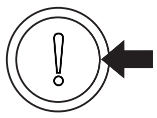
A triangle indicates that there is danger to living things.
The color of the border shows how severe the hazard is: the darker the color, the more severe the hazard is.



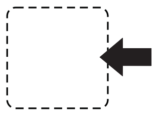
The icon in the rectangle represents the hazard.
The color of the border shows how severe the hazard is: the darker the color, the more severe the hazard is.



The icon in the circle represents an instruction. Users must follow this instruction.
(The circle is shown dashed, since an instruction is not available as an icon for each hazard advisory).



The circle shows that there is a risk of damage to property.



The icon in the rectangle represents the hazard.
The color of the border shows how severe the hazard is: the darker the color, the more severe the hazard is. (The rectangle is shown dashed, since the danger is not represented as an icon with every hazard advisory)

The text next to the icons is structured as follows:

THE SIGNAL WORD IS HERE THAT SHOWS THE DEGREE OF RISK

Here we indicate whether one or more of the results below occurs if you do not observe this warning.

- Here, we describe the possible results. The worst result is always at the extreme right.

Here, we describe the hazard.

Here, we describe what you can do to avoid the hazard.

2.1.2 Hazard advisories that are used

If a signal word is preceded by one of the following danger signs: ⚠ or ⚠ or ⚠, the safety information refers to injury to people.

If a signal word is preceded by a round danger sign: (!), the safety information refers to damage to property.

2.1.2.1 Hazard advisories about injuries to people

To be able to differentiate visually, we use a separate border for each class of hazard information with the triangular and rectangular pictograms.

For danger classification **DANGER,** we use the ⚠ danger sign. The following hazard information of this danger classification is used in this documentation.

DANGER

The following **will occur**, if you do not observe this danger information:

- serious personal injury
- death

*Danger from: **electricity.** The hazard may be described in more detail here.*

Here, we describe what you can do to avoid the hazard.

DANGER

The following **will occur**, if you do not observe this danger information:

- serious personal injury
- death

*Danger from: **mechanical effects.** The hazard may be described in more detail here.*

Here, we describe what you can do to avoid the hazard.

For danger classification **WARNING,** we use the ⚠ danger sign. The following hazard information of this danger classification is used in this documentation.

WARNING

The following **may occur**, if you do not observe this warning information:

- serious personal injury
- death

*Danger from: **electricity.**The hazard may be described in more detail here.*

Here, we describe what you can do to avoid the hazard.

For danger classification **CAUTION,** we use the ⚠ danger sign. The following hazard information of this danger classification is used in this documentation.

CAUTION

The following **may occur**, if you do not observe this caution information:

- minor to medium personal injury.

Danger from: ***sharp edges.*** *The hazard may be described in more detail here.*

Here, we describe what you can do to avoid the hazard.

CAUTION

The following **may occur**, if you do not observe this danger information:

- environmental pollution.

Danger from: ***incorrect disposal.*** *The hazard may be described in more detail here.*

Here, we describe what you can do to avoid the hazard.

2.1.2.2 Hazard advisories about damage to property

If a signal word is preceded by a round danger sign: ⓘ, the safety information refers to damage to property.

CAUTION

The following **may occur**, if you do not observe this caution information:

- property damage.

Danger from: ***electrostatic discharge.*** *The hazard may be described in more detail here.*

Here, we describe what you can do to avoid the hazard.

2.1.2.3 Instruction signs that are used

wear safety gloves

wear safety shoes

2.2 Information signs

NOTE

This indicates particularly important information.

2.3 Legal information

This documentation is intended for technically qualified personnel that has been specially trained and is completely familiar with all warnings and maintenance measures.

The equipment is manufactured to the state of the art and is safe in operation. It can be put into operation and function without problems if you ensure that the information in the documentation is complied with.

Operators are responsible for carrying out servicing and commissioning in accordance with the safety regulations, applicable standards and any and all other relevant national or local regulations with regard to cable rating and protection, grounding, isolators, overcurrent protection, etc.

Operators are legally responsible for any damage that occurs during assembly or connection.

2.4 Appropriate Use

You must always use the plug-in module appropriately. Some important information is listed below. The information below should give you an idea of what is meant by appropriate use of the plug-in module. The information below has no claim to being complete; always observe all the information that is given in these operating instructions.

- You must only install the plug-in module in series b maXX 4400 units.
- Configure the application such that the plug-in module is always operating within its specifications.
- Ensure that only qualified personnel works with this plug-in module.
- Mount the plug-in module only in the specified slot/slots.
- Install the plug-in module as specified in this documentation.
- Ensure that connections always comply with the stipulated specifications.
- Operate the plug-in module only when it is in technically perfect condition.
- Always operate the plug-in module in an environment that is specified in the technical data.
- Always operate the plug-in module in a standard condition.
  For safety reasons, you must not make any changes to the plug-in module.
- Observe all the information on this topic if you intend to store the plug-in module.

You will be using the plug-in module in an appropriate way if you observe all the comments and information in these operating instructions.

2.5 Inappropriate Use

Below, we will list some examples of inappropriate use. The information below should give you an idea of what is meant by inappropriate use of the plug-in module. We cannon, however, list all possible cases of inappropriate use here. Any and all applications in which you ignore the information in this documentation are inappropriate; particularly, in the following cases:

- You installed the plug-in module in units that are not Series b maXX 4400.
- You ignored information in these operating instructions.
- You did not use the plug-in module as intended.
- You handled the plug-in module as follows
  - you mounted it incorrectly,
  - you connected it incorrectly,
  - you commissioned it incorrectly,
  - you operated it incorrectly,
  - you allowed non-qualified or insufficiently qualified personnel to mount the module, commission it and operate it,
  - you overloaded it,
  - You operated the module
    - with defective safety devices,
    - with incorrectly mounted guards or without guards at all,
    - with non-functional safety devices and guards
    - outside the specified environmental operating conditions
- You modified the plug-in module without written permission from Baumüller Nürnberg GmbH.
- You ignored the maintenance instructions in the component descriptions.
- You incorrectly combined the plug-in module with third-party products.
- You combined the drive system with faulty and/or incorrectly documented third-party products.
- Your self-written PLC software contains programming errors that lead to a malfunction.

Version 1.1 of Baumüller Nürnberg GmbH's General Conditions of Sale and Conditions of Delivery dated 2/15/02 or the respective latest version applies in all cases. These will have been available to you since the conclusion of the contract at the latest.

2.6 Protective equipment

In transit, the plug-in modules are protected by their packaging. Do not remove the plug-in module from its packaging until just before you intend to mount it.

The cover on the b maXX units' controller sections provides IP20 protection to the plug-in modules from dirt and damage due to static discharges from contact. This means that you must replace the cover after successfully mounting the plug-in module.

2.7 Personnel training

WARNING

The following **may occur**, if you do not observe this warning information:

- serious personal injury
- death

Only qualified personnel are allowed to mount, install, operate and maintain equipment made by Baumüller Nürnberg GmbH.

Qualified personnel (specialists) are defined as follows:

Qualified Personnel

Electrical engineers and electricians of the customer or of third parties who are authorized by Baumüller Nürnberg GmbH and who have been trained in installing and commissioning Baumüller drive systems and who are authorized to commission, ground and mark circuits and equipment in accordance with recognized safety standards.

Qualified personnel has been trained or instructed in accordance with recognized safety standards in the care and use of appropriate safety equipment.

Requirements of the operating staff

The drive system may only be operated by persons who have been trained and are authorized.

Only trained personnel are allowed to eliminate disturbances, carry out preventive maintenance, cleaning, maintenance and to replace parts. These persons must be familiar with the Operating Instructions and act in accordance with them.

Commissioning and instruction must only be carried out by qualified personnel.

2.8 Safety measures in normal operation

- At the unit's place of installation, observe the applicable safety regulations for the plant in which this unit is installed.
- Provide the unit with additional monitoring and protective equipment if the safety regulations demand this.
- Observe the safety measures for the unit in which the plug-in module is installed.

2.9 Responsibility and liability

To be able to work with these option modules in accordance with the safety requirements, you must be familiar with and observe the hazard information and safety instructions in this documentation.

2.9.1 Observing the hazard information and safety instructions

In these operating instructions, we use visually consistent safety instructions that are intended to prevent injury to people or damage to property.

WARNING

The following **may occur**, if you do not observe this warning information:

- serious personal injury
- death

Any and all persons who work on and with Series b maXX units must always have available these Operating Instructions and must observe the instructions and information they contain – this applies in particular to the safety instructions.

Apart from this, any and all persons who work on this unit must be familiar with and observe all the rules and regulations that apply at the place of use.

2.9.2 Danger arising from using this module

The option module has been developed and manufactured to the state of the art and complies with applicable guidelines and standards. It is still possible that hazards can arise during use. For an overview of possible hazards, refer to the chapter entitled ▷Basic Safety Instructions ◁ from page 7 onward..
We will also warn you of acute hazards at the appropriate locations in this documentation.

2.9.3 Warranty and Liability

All the information in this documentation is non-binding customer information; it is subject to ongoing further development and is updated on a continuous basis by our permanent change management system.

Warranty and liability claims against Baumüller Nürnberg GmbH are excluded; this applies in particular if one or more of the causes listed in ▷Inappropriate Use ◁ from page 12 onward or below caused the fault:

- Disaster due to the influence of foreign bodies or force majeure.

3

CANopen

This chapter contains information on data exchange via CANopen.

The option module BM4-O-CAN-03 is available as CANopen Slave.

The option modules BM4-O-ETH-02 and BM4-O-CAN-04 are available as CANopen Master.

3.1 General information on CANopen and use of the CANopen Slave option module

CANopen is a widespread field bus application layer, which is based on the Controller Area Network (CAN) bus system and has been published by the international CAN organization CAN in Automation e.V. (CiA).

The CANopen mechanisms and functionalities are described by different profiles. The communication profile defines the method of data exchange and general definitions applicable for all CANopen devices. Device profiles contain application- and device-specific definitions describing the contents-related meaning of data and device functionality. Device profiles exist for drives, I/O modules, encoders and programmable devices, etc..

With the CANopen Slave option module for the b maXX PLC, you can implement a data exchange with slave functionalities according to communication profile DS 301, together with the function blocks of library CANop405_PLC01_20bd03 (or higher) for the programming environment PROPROG wt II.

The programming uses the function blocks from the library CANop405_PLC01_20bd03 (or higher) under PROPROG wt II. The function blocks are implemented according to the device profile DS 405. This means that CANopen and CANopen Slave functions such as

- Process data exchange via process data objects (PDOs) with high priority identifiers
- Requirements data exchange via service data objects (SDOs) with low priority identifiers
- Synchronization (SYNC) of real time data exchange with PDOs and SYNC monitoring
- Error (EMERGENCY): Set emergency send job (Slave --> Master)
- Network monitoring (NODE GUARDING / HEARTBEAT): for failure monitoring of network nodes

can be easily implemented.

In addition, network management commands (NMT commands) such as e.g. Start, Stop and ResetCommunication are executed.

The five transfer rates 1 MBit/s, 500 kBit/s, 250 kBit/s, 125 kBit/s, and 20 kBit/s are available for the CANopen Slave. The CANopen Master option module can communicate with up to 32 CANopen network nodes (e. g. I/O modules and drives).

3.2 Basic principles of CAN and CANopen networks

3.2.1 Basic principles of CAN

A CAN-based field bus system is executed in line structure. A three-wire cable provides the physical basis of data transfer with the connections CAN_High, CAN_Low and CAN_Ground. CAN employs transmissions balanced to ground for the suppression of common mode interference. For this reason, the difference signals are evaluated.

Network

CAN is a multi-master network. Every participant can have access to and be active on the bus with equal priority. CAN uses object-oriented addressing, i.e. the transferred message is identified by an identifier defined network-wide. The identifier represents the encoded name of the message.

Bus access

Bus access takes place using the CSMA/CA procedure (Carrier Sense Multiple Access / Collision Avoidance). Because after detecting the necessary bus quiescence every participant has the right to begin transmitting a message, collisions can occur. This is avoided by means of bit-by-bit arbitration of the messages to be sent. During this, two bus levels are differentiated, a dominant level, logical bit value 0 and a recessive level, logical bit value 1. In the worst case, all participants wishing to transmit simultaneously begin sending their messages onto the bus. If a recessive bit of a participant is overwritten by a dominant bit of another, then the "recessive" node withdraws from the bus and, after detecting bus quiescence, attempts once more to transmit its message. Consequently, it is guaranteed that the most important and highest priority message (with the lowest identifier) will be transmitted in a collision-free manner and without delay. For this reason it is of course necessary that every identifier is permitted to be placed onto the CAN bus only once.

Identifier

Different identifiers are available in accordance with specification CAN 2.0A 2032. Each participant can transmit in an unsolicited manner (multi-master capability). A transmitter sends its message to all CAN nodes (broadcast) and each then decides on the basis of the identifier whether they will continue processing the message or not.

Error

Up to 8 bytes of user data can be transmitted within a CAN data message frame. For error or overload signaling, a CAN node can send error or overload message frames. This occurs on Layer 2 of the OSI/ISO reference model, the Data Link layer, therefore independently of the application. Due to high quality error detection and handling on Layer 2, a

Hamming distance (a measure of error detection) of HD=6 is achieved, i. e. a maximum of 5 simultaneously-occurring bit errors within a message frame will be reliably detected as an error.

3.2.2 Basic principles of CANopen

CANopen is an open and hence manufacturer-independent field bus system defining Layer 1 and 2 of the CAN standard.

3.2.2.1 CAL specification and profiles

In the International CAN Organization CAN in Automation e.V. (CiA), a general description language was created for CAN networks with the CAL application layer. CAL provides a collection of communication services without precisely defining their application. CANopen uses this description language. On the one hand, part of the communication services offered by CAL is used to define an open communication interface. These definitions are recorded in the communication profiles. Profile DS 301, in which the ’how’ of the communication is regulated, is particularly important. On the other hand, CANopen also defines what the data in the respective device classes mean.

These definitions are recorded in the device profiles, for I/Os this is e. g. profile DS 401 and for drives profile DSP 402. Objects in devices programmable according to IEC 61131-3 are defined by the device profile DS 405.

The CANopen Slave option module, together with the function blocks from the library CANop405_PLC01_20bd03 (and higher) for PROPROG wt II, supports the communication profile DS 301 and the device profile DS 405.

3.2.2.2 Communication and device-specific objects

When access is made via the network, the objects of a network node are accessed. Objects undertake different tasks. Data exchange is configured via the CANopen field bus through communication profile-specific objects. As device profile specific objects, these are directly connected to the device. The device functions can be used and changed through these. Objects are always addressed via an index (16 bit) and a subindex (8 bit). Depending on the importance, various object areas are defined. The most important user areas are:

Index (hex)	Object
1000 - 1FFF	Range for the communication profile. Contains all objects that are required for communication, e.g. PDO, SDO.
2000 - 5FFF	Range for manufacturer-specific objects. Contains all objects that are not defined within a profile and are manufacturer-dependent.
6000 - 9FFF	Range for the device profile. Contains the objects of the supported device profile (DS 401, DSP402, etc.)

Which objects must be present and which are optional for a network node is precisely defined in the individual profiles.

3.2.2.3 Data exchange and objects of the physical bus system

From the point of view of the communication relationships between the CANopen participants, the data exchange is described by the

- Client - Server model: each is entitled to request data from another participant or to transfer data to it

and the

- Producer - Consumer model: a participant sends data without a request, others listen in and can evaluate the data.

No master - slave relationship (one CANopen participant organizes the data exchange) exists on the communication relationship level.

A CANopen Slave does not therefore mean a slave on the communication level, but on the application level. CANopen Slave functions therefore mainly comprise tasks such as status queries, detection and processing of requests.

Data exchange in CANopen networks occurs in the form of message frames, which are used to transfer the user data and access the objects of a network node. Each message frame is identified by an identifier. The message frame types and the basic mechanisms for data exchange are divided into groups:

- Process data objects (PDOs): Real time data exchange with high-priority identifiers and up to 8 bytes per message. The CANopen Slave option module can receive a maximum of 4 PDOs and send a maximum of 4 PDOs.
- Service data objects (SDOs): Parameter data exchange with low priority identifiers and through index/subindex addressable data. The transfer type for the CANopen Slave option module is "expedited", i. e. up to 4 bytes can be transferred per message. The CANopen Slave option module can receive or send a maximum of 1 SDOs simultaneously.
- Network management (NMT) Special object type for implementation of network communication functions such as e.g. Start, Stop and ResetCommunication of network nodes. The CANopen Slave option module converts the commands sent by a CANopen Master.
- Synchronization (SYNC): Special object type for synchronization of real time data exchange with PDOs. The CANopen Slave option module synchronizes its data exchange (via PDO) with the SYNC command sent by a CANopen Master.
- Error handling (EMERGENCY): Special object type for the detection of errors in a network node. The CANopen Slave option module sends an emergency message frame in the event of an error.
- Network monitoring (NODE GUARDING): Special object type for failure monitoring of network nodes. The CANopen Slave option module cyclically sends a message frame upon request of the CANopen Master. It also monitors the request of this message frame and thus detects failure of the NMT Master (the CANopen Master).

NOTE

In the profile definitions of the CiA, the term "Object" is used in two different ways. On the one hand, as in Chapter ▷Communication and device-specific objects ◁ on page 17, as a type of datum which can be accessed externally, or which describes device properties. On the other hand, it also designates the different message frame types and the associated mechanisms for data exchange. In the second case, these objects can be seen as objects of the physical CANopen bus system, but not as objects which can be accessed via indices and subindices. These objects of the physical CANopen bus system transport the communication and device-specific objects.

These definitions also mean that, in contrast to other field bus systems, the message frames themselves are identified by the individual participants. Each network node decides for itself when it wishes to send data and which message frames it evaluates. It is, of course, also possible to prompt other network nodes to send data via so-called remote message frames.

3.2.2.4 Predefined identifier

To enable establishment of peer-to-peer communication between the master and the other network nodes (slaves) directly after a boot-up, there is a predefined identifier assignment for the message frames. This identifier assignment can be reconfigured by the user. An identifier (according to CAN definition) is divided into function code and module ID:

Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Function code				Module ID						

The module ID is synonymous with the node number in the network. From the 7 bits for the module IDs, a maximum number of 127 nodes results per CANopen network. For physical reasons, a maximum of 32 nodes can be operated in one network with the CANopen Slave option module. The COB ID (Communication Object Identifier) results from the total of 11 bits. The following objects are predefined:

Broadcast objects (Module ID = 0):

Object name	**Binary function code**	**Resultant COB ID decimal**	**Resultant COB ID hexadecimal**
NMT	0000	0	16#0
SYNC	0001	128	16#80
TIME STAMP [a)]	0010	256	16#100

a) The Time Stamp object is not supported by the CANopen Master option module.

Peer-to-peer objects:

Object name	Binary function code	Resultant COB ID decimal	Resultant COB ID hexadecimal
EMERGENCY	0001	129 - 255	81h - FFh
PDO1 (TX)	0011	385 - 511	181h - 1FFh
PDO1 (RX)	0100	513 - 639	201h - 27Fh
PDO2 (TX)	0101	641 - 767	281h - 2FFh
PDO2 (RX)	0110	769 - 895	301h - 37Fh
PDO3 (TX)	0111	897 - 1023	381h - 3FFh
PDO3 (RX)	1000	1025 - 1151	401h - 47Fh
PDO4 (TX)	1001	1153 - 1279	481h - 4FFh
PDO4 (RX)	1010	1281 - 1407	501h - 57Fh
SDO (TX)	1011	1409 - 1535	581h - 5FFh
SDO (RX)	1100	1537 - 1663	601h - 67Fh
NODE GUARDING	1110	1793 - 1919	701h - 77Fh

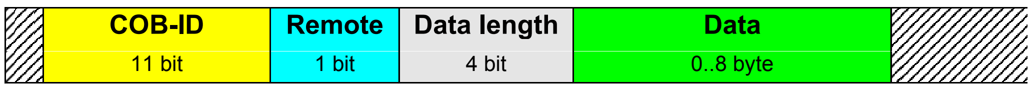
TX stands for transmit (send) and RX for receive. Please note that the transfer direction is viewed from the network node, not from the master. I. e. if for example a PDO is sent from the CANopen Master , this is an RxPDO for the CANopen Slave option module and must therefore be transferred to the CANopen Bus with an RX identifier. The same applies for the module ID, which is also viewed from the CANopen Slave option module and not from CANopen Master, i. e. the COB ID contains the module ID of the CANopen Slave and not the CANopen Master.

However, these conditions only apply for the predefined identifier assignment. According to the DS 301 communication profile, the COB IDs for the bus physical communication objects can be freely defined by the user for each network node via device objects. Therefore, e. g. a process data communication can be set up directly between two (or more) network nodes (e. g. between two b maXX 4400 devices with b maXX PLC and CANopen Slave option module).

3.2.2.5 Message frame structure and priority

In the CAN message frame, the most important CANopen elements are the COB ID, the remote bit, the data length and the data. As the COB ID corresponds to the CAN identifier, on the basis of the CSMA/CA CAN transfer procedure (see Chapter ▷Basic principles of CAN ◁ on page 16), low value COB IDs have a higher priority. The predefined identifier assignment thus means that NMT message frames have the highest priority, followed by SYNC message frames. As the COB ID also contains the network node number, message frames of network nodes with a lower node number also have higher priority.

The most important elements in the CANopen message frame are:



If the remote bit is set, data are requested from other network nodes. Up to 8 bytes are available for data. However, the number of bytes is dependent on the message frame type. With SDO this is a maximum of 4 bytes, the remaining 4 bytes are needed to identify the data type. PDOs can use all 8 bytes. SYNC message frames have no data contents.

3.3 Data exchange via CANopen with the CANopen Slave option module

3.3.1 General information on data exchange

The following chapters explain how a CANopen Slave option module is used in a CANopen network.

They highlight both the basic mechanisms as defined in profile DS 301 and the use of the function blocks from the library CANop405_PLC01_20bd03 (or higher) according to profile DS 405. The representations are accompanied by an example project for PROPROG wt II, generated step-by-step.

For commissioning the CANopen Master, please refer to the relevant Operating Instructions.

NOTE

The resulting example project should by no means be seen as a fully functional application. The values of the data that are written via the examples from the CANopen Slave to the b maXX PLC and, if applicable, also to the b maXX controller, have no functional meaning. Therefore, please make sure that the drive (motor) is not in operational status!

Naturally, the CANopen Slave and the function blocks offer far more possibilities than can be described in the scope of this manual. It is also not possible to go into all the details of the function blocks. You can obtain detailed information on the function blocks in the relevant online help in PROPROG wt II.

3.3.2 Overview of data exchange

Access to the data of a CANopen Slave always occurs via the objects. In the CANopen Slave option module for b maXX PLC, access to the data occurs via the profile-specific objects from the CiA device profile DS 405. These are objects 0xA000 to 0xA8FF.

How the respective data access occurs and further general information is defined in the communication-specific objects. These are objects 0x1000 to 0x1FFF.

Although the b maXX 4400 with b maXX PLC and CANopen Slave option module is a drive, DS 405 (IEC 61131-3 Programmable Devices) is supported as CiA device profile, not DSP 402 (Drives).

Reason: The CANopen Master does not directly access the drive. The CANopen Master accesses the data of the b maXX PLC (which is programmed according to IEC 61131-3).

The b maXX 4400, as CANopen Slave, receives the data from the CANopen Master. In the b maXX 4400, the CANopen Slave option module receives the data and forwards this to the b maXX PLC. The data can be further processed in the b maXX PLC (e. g. with an electronic cam disk) and are then transferred to the b maXX controller. The actual regulation and control of the drive (motor) is performed here by the b maXX power section.

Vice-versa, the data can be transmitted to the CANopen Master.

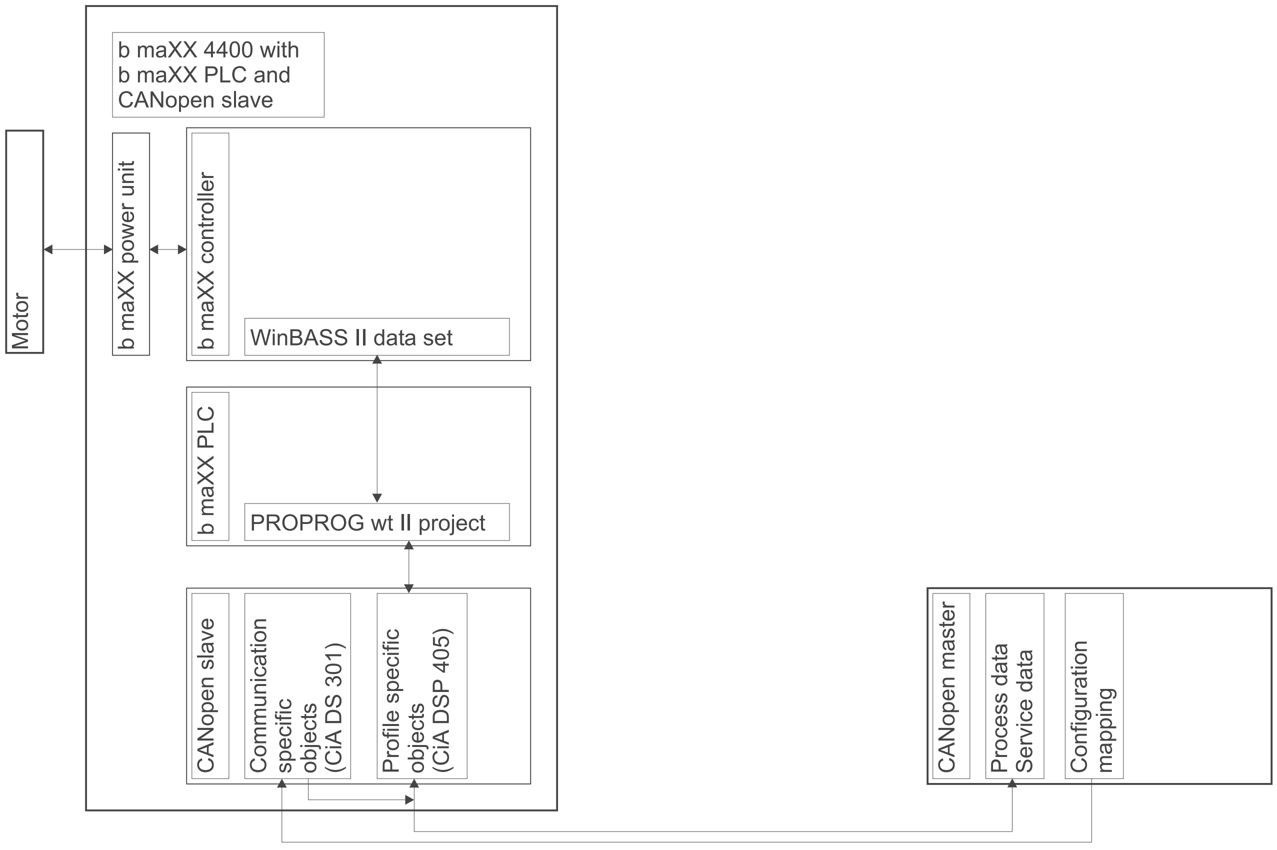


Figure 1: Data flow CANopen Master → CANopen Slave → b maXX PLC → b maXX controller → Power unit → Motor and back

Communication CANopen Master → CANopen Slave

The CANopen Master communicates with the CANopen Slave via the process data objects (PDOs) and service data objects (SDOs). The contents of PDOs and SDOs are the profile-specific objects (according to profile DS 405).

CANopen Slave

On the CANopen Slave, the PDOs and SDOs are converted to the corresponding objects in the CANopen Slave. A mapping is stored on the CANopen Slave during initialization for this purpose. This mapping is created by the CANopen Slave configurator in the b maXX drive configurator during the creation of variables.

Communication CANopen Slave → b maXX PLC

These profile-specific objects are assigned via the CANopen Slave configurator (in the b maXX drive configurator; see ▷Creating variables ◁ from page 34 onward) to the variables in the PROPROG wt II project on the b maXX PLC.

b maXX PLC

In the b maXX PLC, the data is processed further as a rule (using the variables) e. g. by means of electronic cam. The reference values resulting thereby are transferred to the b maXX controller. The programming of the application program in the b maXX PLC takes place via PROPROG wt II (see Manual PROPROG wt II and Application Manual b maXX PLC).

Communication b maXX PLC → b maXX controller

The reference values are transfered by the b maXX PLC to the b maXX controller. In the b maXX controller, these reference values are then the parameter like position setpoint. The programming of the communication between the b maXX PLC and the b maXX controller takes place via PROPROG wt II. The programming of the communication is described in the Application Manual for the b maXX PLC and is not explained in detail in this document.

b maXX controller

In b maXX controller, the control loop is carried out by using the reference values (of the b maXX PLC) and the actual values of the drive (or the b maXX power unit). The conditions for the control loop (e. g. torque limit) can be parametrised.

The parametrisation of the b maXX controller is carried out by using WinBASS II (see Parameter Manual for b maXX 4400).

Communication b maXX controller → b maXX power unit

The set values of the parameters of the b maXX controller are transferred to the b maXX power unit.

b maXX power unit / motor

In b maXX power unit, the set values of the parameters of the b maXX controller are converted in current and voltage, at which the motor is run. Actual values (e. g. actual value of the current) are also determined in b maXX power unit. Other actual values (e. g. actual values for the motor speed or the actual position) are also determined by the encoder on the motor and the encoder module on the b maXX controller.

The actual values can be reported in the reverse way, to the CANopen Master.

3.3.3 Steps to be executed

Assembly, connection and the commissioning of

- Drive
- b maXX 4400
- Option Module b maXX PLC (BM4-O-PLC-01)
- Option Module CANopen Slave (BM4-O-CAN-03)
- as also of the CANopen Master used by you

must have been completed successfully.

The following steps are required to enable data exchange with the CANopen Slave via the CANopen network:

- Creation of a PROPROG wt II project for the b maXX PLC.
- Device configuration with the b maXX configurator.
- Configuration of the CANopen Slave option module with the CANopen Slave configurator (in the b maXX configurator).
  - Result:
    List of variables for SDOs and PDOs for the PROPROG wt II project,
    EDS file for a CANopen Master configuration tool
- Programming of the application section in the PROPROG wt II project for the b maXX PLC.
  - Result:
    PROPROG wt II example project with INIT, NMT, EMCY, SDO, PDO.
- If required, integration of the EDS file into a CANopen Master configuration tool.

3.3.4 Creating a PROPROG wt II project

3.3.4.1 Procedure for creating a project

In order to be able to use the CANopen Master option module with the CANopen function blocks in accordance with profile DS 405, you need a PROPROG wt II project for the b maXX PLC. If you have not yet created a project for your application, please create the project with the template *bmaXX4000 with configurator*. To do this, you need a PROPROG wt II Version 3.1 from Build 274 and the b maXX Drive-Configurator for PROPROG wt II Version 1.0 (or higher). You can find the version number of PROPROG wt II on the envelope of the installation CD for PROPROG wt II or in PROPROG wt II itself in menu item **? \ Info**. Also check that the library BM_TYPES_20bd06 (or higher) is present in your PROPROG wt II project. If this is not the case, please link this library in your project. It contains important data types for CANopen. Then link the CANop405_PLC01_20bd03 library in your project.

3.3.4.2 Example: Creating the "CANopenSlave_Example" project

The example project "CANopenSlave_Example" was created with the template *bmaXX4000 with Drive-Configurator* and the libraries BM_TYPES_20bd06 and CANop405_PLC01_20bd03 were linked.

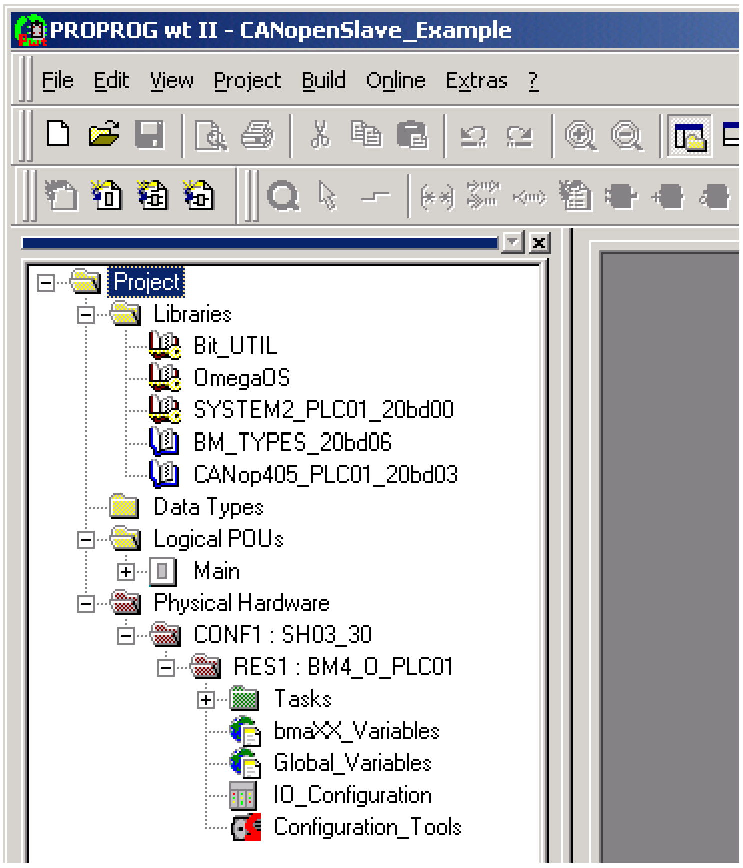


Figure 2: Example: Creating the "CANopenSlave_Example" project

3.3.4.3 Creating a global variable for data exchange

A global variable is required for data exchange between option module BM4-O-CAN-03 and the function blocks. It has no further significance for the user. This global variable is already created in your project, provided that the project was created with the template *bmaXX4000 with drive configurator*. This global variable is connected to the input/output _CANop405_CTRL at the function blocks for CANopen.

Depending on the slot used on the option module (Slots G to M), the global variables _CANopenSlave_Ctrl_Slot_G to _CANopenSlave_Ctrl_Slot_M are available.

You can also find these in the "Global_Variables" worksheet:

```
(* Option module CANopen-Slave (BM4-O-CAN-03) *)
   _CANopenSlave_Ctrl_Slot_G     AT %MB 3.2001792  :   CANopen_PLC_SL_BMSTRUCT;
   _CANopenSlave_Ctrl_Slot_H     AT %MB 3.3001792  :   CANopen_PLC_SL_BMSTRUCT;
   _CANopenSlave_Ctrl_Slot_J     AT %MB 3.4001792  :   CANopen_PLC_SL_BMSTRUCT;
   _CANopenSlave_Ctrl_Slot_K     AT %MB 3.5001792  :   CANopen_PLC_SL_BMSTRUCT;
   _CANopenSlave_Ctrl_Slot_L     AT %MB 3.6001792  :   CANopen_PLC_SL_BMSTRUCT;
   _CANopenSlave_Ctrl_Slot_M     AT %MB 3.7001792  :   CANopen_PLC_SL_BMSTRUCT;
```

Figure 3: Global variables for the CANopen Slave option module, depending on the slot

If the global variable _CANopenSlave_Ctrl_Slot_G (upto _CANopenSlave_Ctrl_Slot_M) of the data type CANop405_PLC_SL_BMSTRUCT required for the slot of the Option Module is not present in the project, then create it, depending on the slot (Slot G to M). This variable must be declared as the global variable and placed at the base address for the CANopen Slave communication of the Option Module BM4-O-CAN-03 The base address depends on the slot:

Slot	Base address for CANopen Master communication
G	%MB3.2001792
H	%MB3.3001792
J	%MB3.4001792
K	%MB3.5001792
L	%MB3.6001792
M	%MB3.7001792

NOTE

The variable name is replaced by * in the illustration below. Furthermore, it is assumed that the Option Module CANopen Slave for b maXX PLC has been plugged in the slot G.

As a result, the register *.u_BAUDRATE is accessed as follows:

```
_CANopenSlave_Ctrl_Slot_G.u_BAUDRATE  .
```

Where:

`_CANopenSlave_Ctrl_Slot_G`	the variable name with the data type short designation "_" for Struct
`u_BAUDRATE`	the register in which the baud rate is entered, at which the CANopen Slave option module operates on the CAN bus.

3.3.5 Device configuration with the b maXX Configurator

The PROPROG wt II project, which you have already created (under ▷Creating a PROPROG wt II project ◁ on page 24), contains the button for the configurators (Configuration_Tools). After double-clicking on these buttons, the configurators window gets opened. This contains the b maXX configurator amongst others (Drive_Configuration).

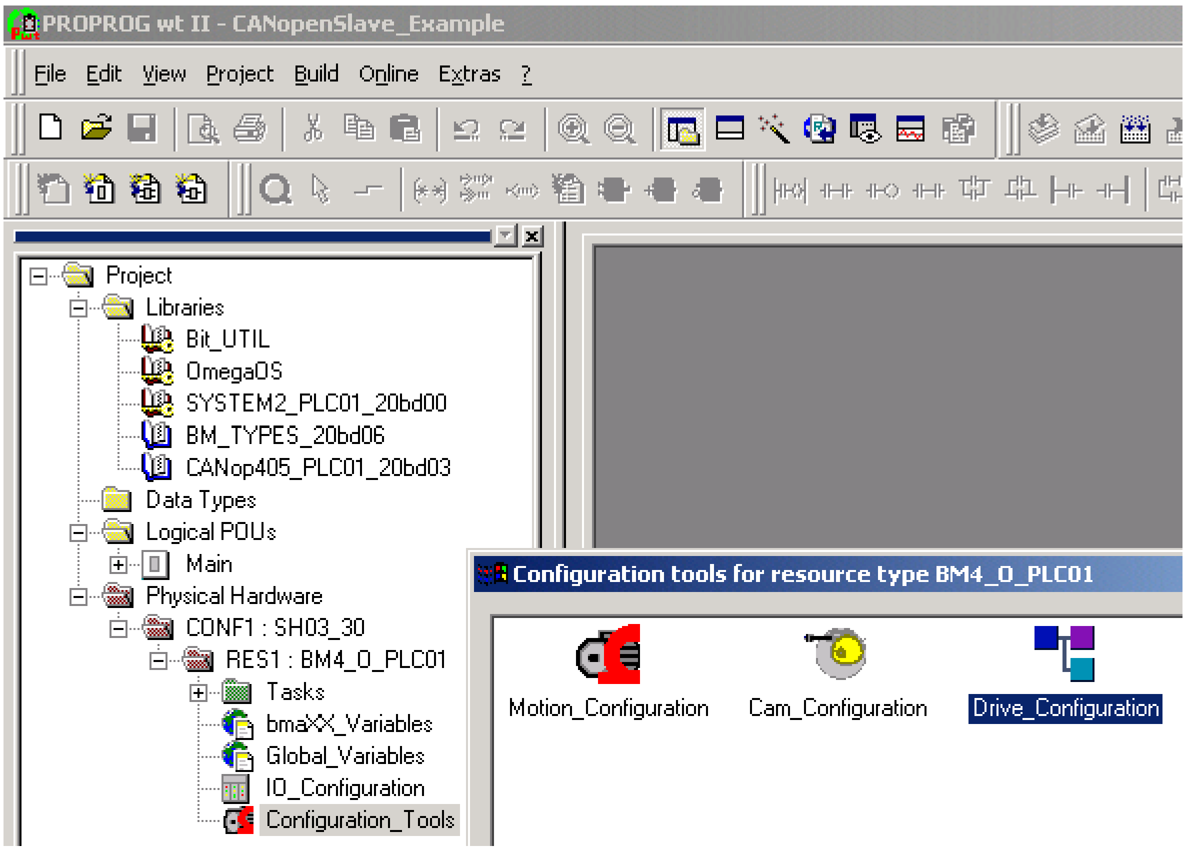
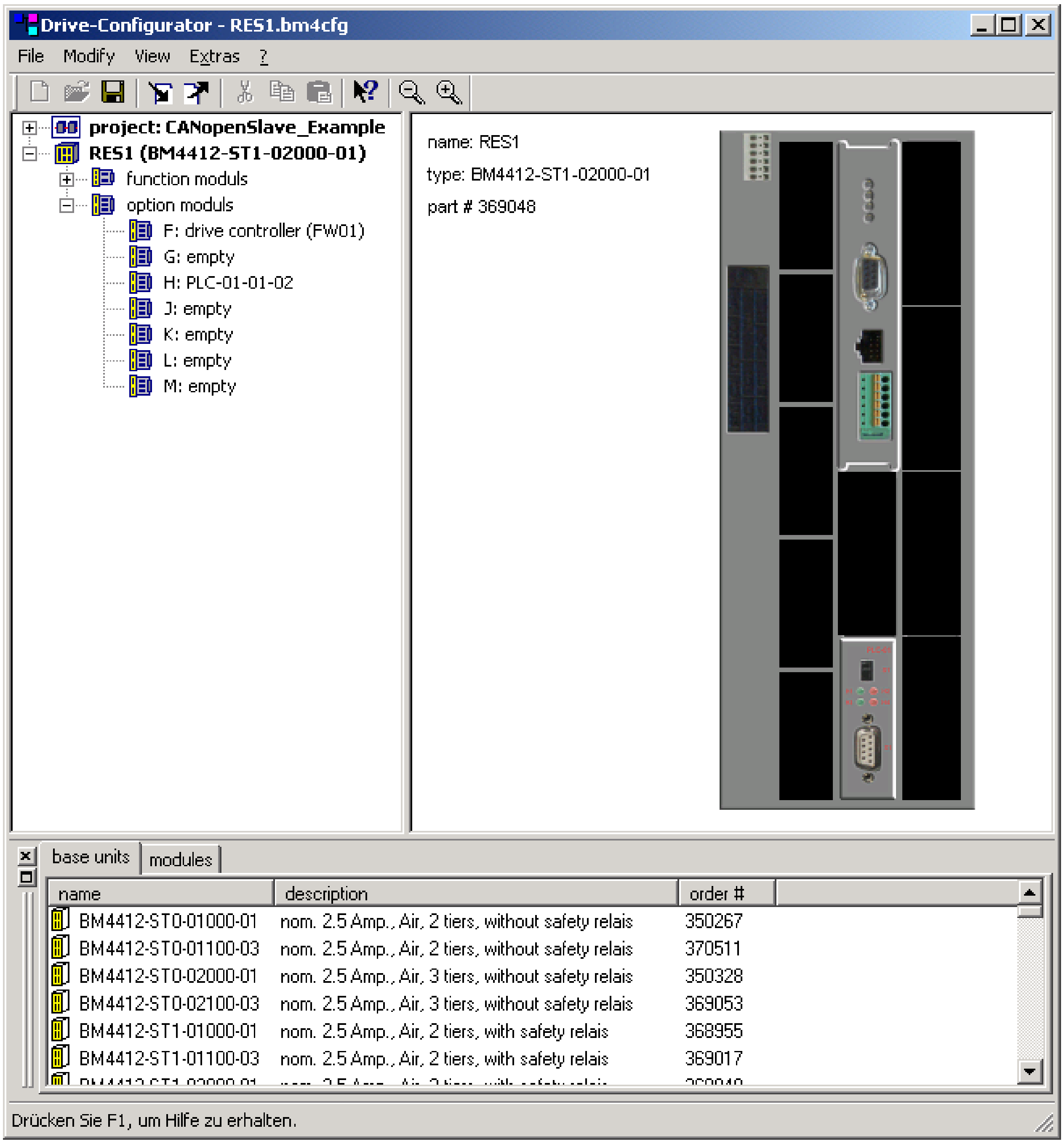


Figure 4: Opening the b maXX Configurator

Before opening the b maXX configurator, please compile your PROPROG wt II project (Menu: Code\Create new project).

Now click on the button „Drive_Configuration“ to open the b maXX Configurator.



project: CANopenSlave_Example
RES1 (BM4412-ST1-02000-01)
function moduls
option moduls
F: drive controller (FW01)
G: empty
H: PLC-01-01-02
J: empty
K: empty
L: empty
M: empty

name: RES1
type: BM4412-ST1-02000-01
part # 369048

base units | modules

name	description	order #
BM4412-ST0-01000-01	nom. 2.5 Amp., Air, 2 tiers, without safety relais	350267
BM4412-ST0-01100-03	nom. 2.5 Amp., Air, 2 tiers, without safety relais	370511
BM4412-ST0-02000-01	nom. 2.5 Amp., Air, 3 tiers, without safety relais	350328
BM4412-ST0-02100-03	nom. 2.5 Amp., Air, 3 tiers, without safety relais	369053
BM4412-ST1-01000-01	nom. 2.5 Amp., Air, 2 tiers, with safety relais	368955
BM4412-ST1-01100-03	nom. 2.5 Amp., Air, 2 tiers, with safety relais	369017

Drücken Sie F1, um Hilfe zu erhalten.

Figure 5: b maXX configurator

Description of the steps:

- Selection of b maXX 4000 basic unit (including b maXX controller)
- Selection of b maXX PLC option module
- Selection of CANopen Slave option module

3.3.5.1 Selection of b maXX 4000 basic unit (including b maXX controller)

In the project tree of the b maXX Configurator, a b maXX basic unit including b maXX controller is set as default. In Figure 18, the b maXX basic unit BM4412-ST1-02000-01 is set under resource "RES1".

- If you wish to use another b maXX basic unit, select "Add/Change" in the resource context menu.
- Select your b maXX basic unit in the "Select basic device type" window.
- Confirm the selection with "OK".

The new b maXX basic unit (including b maXX controller) is now accepted and the graphic adapted.

The b maXX basic unit BM4412-ST0-01000-01 is used in the example project (see ▹Figure 6).

3.3.5.2 Selection of b maXX PLC option module

- Open the entries for the option modules in the project tree of the b maXX Configurator (click on "+" before "Option modules").

A b maXX PLC is set as default in slot "H". In Figure 18, the b maXX PLC BM4-O-PLC-01-01-02 is set in slot "H".

- If you wish to use another b maXX PLC, select "Add/Change" in the context menu for slot "H".
- Select your b maXX PLC in the "Option module for slot H" window.
- Confirm the selection with "OK".

The new b maXX PLC is now accepted and the graphic adapted.

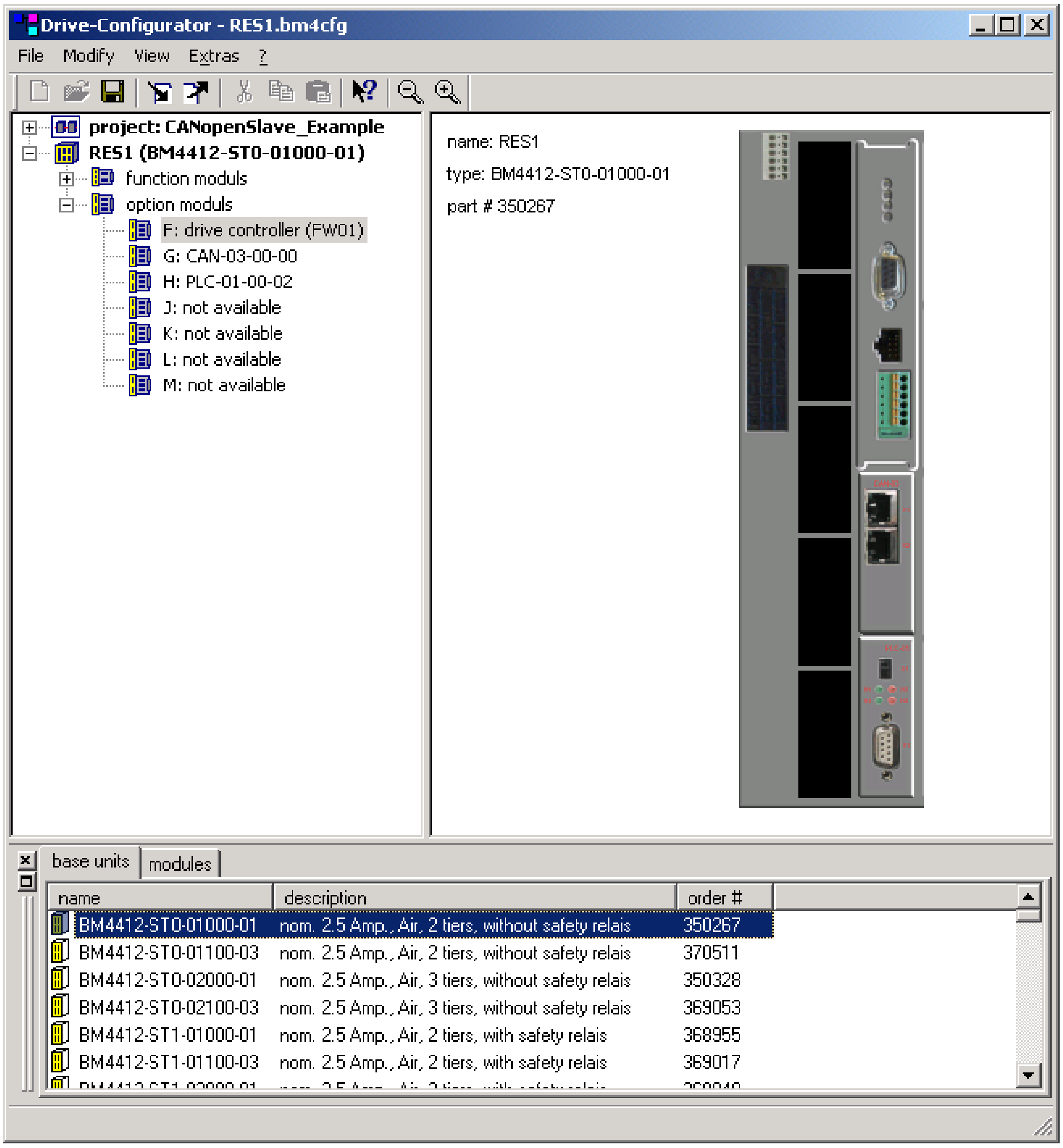
In the example project, the b maXX PLC BM4-O-PLC-01-00-01 is used in slot "H" (see ▹Figure 6).

3.3.5.3 Selection of CANopen Slave option module

- Select "Add/Change" in the context menu for slot "G".
- Select the CANopen Slave option module for b maXX (BM4-O-CAN-03-00-00) in the "Option module for slot G" window.
- Confirm the selection with "OK".

The CANopen slave option module for b maXX PLC is now accepted and the graphic adapted.

In the example project, the CANopen Slave option module for b maXX PLC BM4-O-CAN-03-00-00 is used in slot "G".



project: CANopenSlave_Example
RES1 (BM4412-ST0-01000-01)
function moduls
option moduls
F: drive controller (FW01)
G: CAN-03-00-00
H: PLC-01-00-02
J: not available
K: not available
L: not available
M: not available

name: RES1
type: BM4412-ST0-01000-01
part # 350267

name	description	order #
BM4412-ST0-01000-01	nom. 2.5 Amp., Air, 2 tiers, without safety relais	350267
BM4412-ST0-01100-03	nom. 2.5 Amp., Air, 2 tiers, without safety relais	370511
BM4412-ST0-02000-01	nom. 2.5 Amp., Air, 3 tiers, without safety relais	350328
BM4412-ST0-02100-03	nom. 2.5 Amp., Air, 3 tiers, without safety relais	369053
BM4412-ST1-01000-01	nom. 2.5 Amp., Air, 2 tiers, with safety relais	368955
BM4412-ST1-01100-03	nom. 2.5 Amp., Air, 2 tiers, with safety relais	369017

Figure 6: b maXX Configurator according to device configuration

Continue with "Configuration of the CANopen Slave option module by using the CANopen Slave configurator".

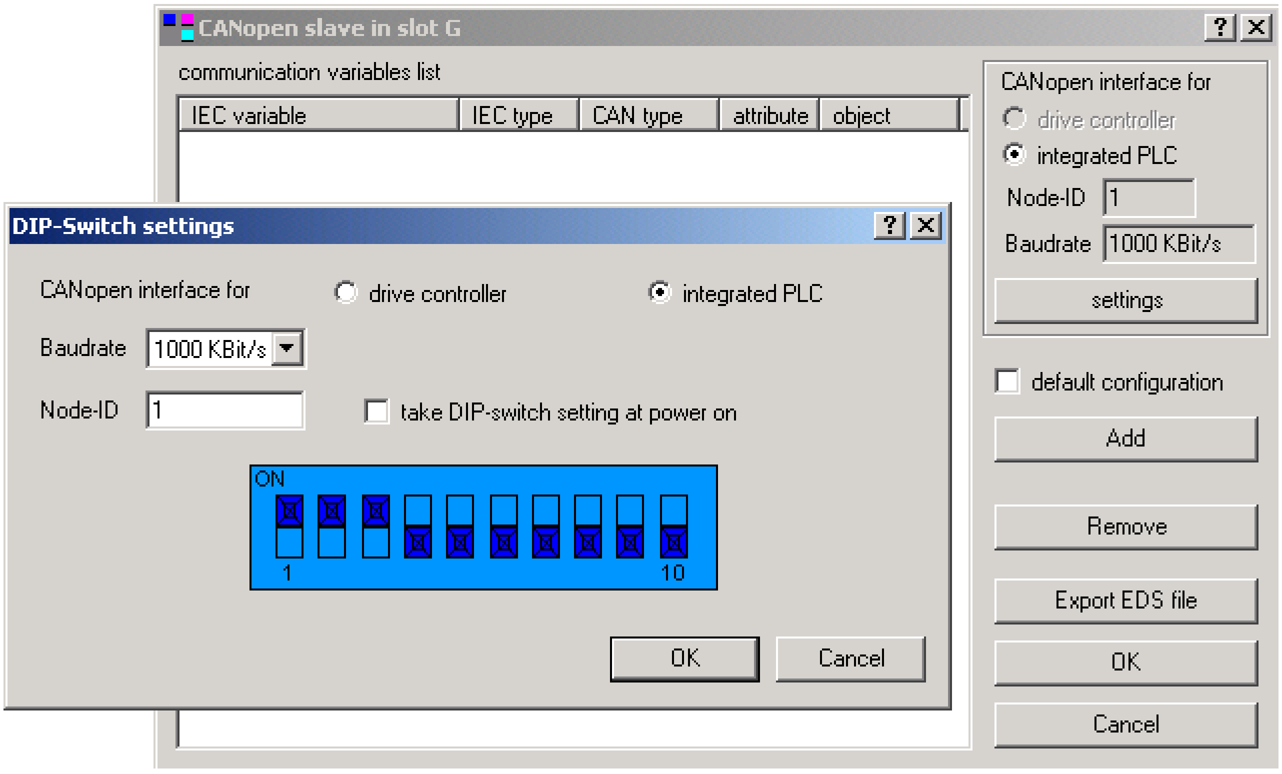
3.3.6 Configuration of the CANopen Slave option module by using the CANopen Slave Configurator

This chapter explains the operation of the CANopen Slave Configurator. The CANopen Slave Configurator

- generates parts of the datasheet of the global variable worksheet "bmaXX_Variables" for the PROPROG wt II project
  (reads in parts from an existing global variables worksheet "bmaXX_Variables" for the PROPROG wt II project beforehand, if applicable)
- generates the EDS file for the key data of the devices describing the device „b maXX 4400 with b maXX PLC and the Option Module CANopen Slave" for the CAN-open Master configurators.

If necessary, open the CANopen Slave Configurator from PROPROG wt II via Configuration_Tools → Drive_Configuration → "Option modules \ G: CAN-03-00-00" - "Configure" in the context menu.

- Click on the "Settings" button in the "CANopen interface for" area in the CANopen Slave Configurator.
- Activate the radio button "CANopen interface for ... integrated PLC"


CANopen slave in slot G

communication variables list

IEC variable	IEC type	CAN type	attribute	object

CANopen interface for: drive controller [✓ integrated PLC]
Node-ID: 1
Baudrate: 1000 KBit/s

default configuration: ☐

DIP-Switch settings

CANopen interface for: drive controller [✓ integrated PLC]
Baudrate: 1000 KBit/s
Node-ID: 1
take DIP-switch setting at power on: ☐

Figure 7: Selection CANopen interface for integrated PLC

By activating the radio button "CANopen interface for ... integrated PLC", the display of DIP switches 1, 2 and 3 is set to "ON". When commissioning the CANopen Slave option module, you must set these DIP switches to "ON", so that your option module is a CANopen Slave option module for b maXX PLC.

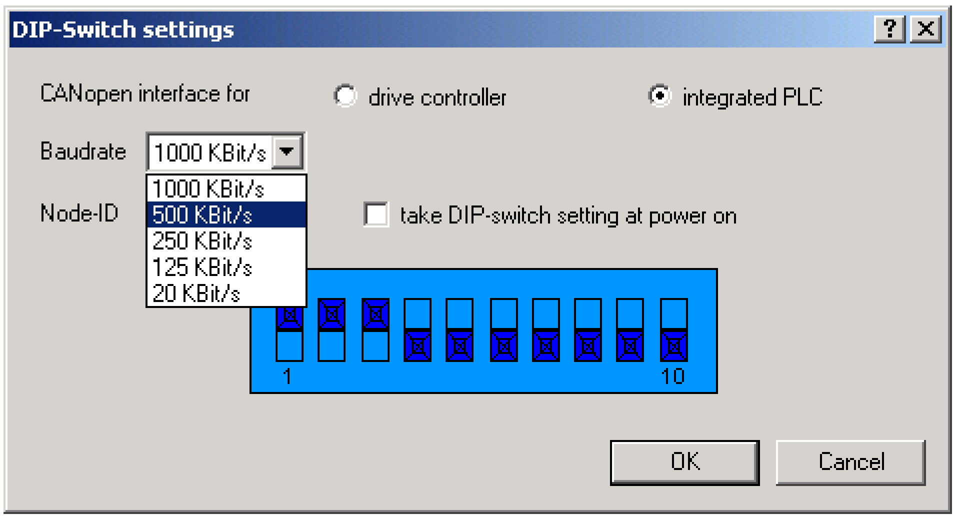
Continue with "Setting the baud rate".

3.3.6.1 Setting the baud rate

Set the baud rate at which the CANopen Slave option module is to operate on the CANopen bus system as follows:

- Click on the "Settings" button in the "CANopen interface for" area in the CANopen Slave Configurator, provided that the "DIP-Switch settings" window has not yet opened.
- Click in the "Baud rate" edit box.
- Select the desired baud rate from the list.

The baud rate 500 kbit/s is selected in the example.



DIP-Switch settings

CANopen interface for ( ) drive controller [✓ (•) integrated PLC]

Baudrate: 1000 KBit/s

1000 KBit/s
[✓ 500 KBit/s]
250 KBit/s
125 KBit/s
20 KBit/s

Node-ID

[ ] take DIP-switch setting at power on

Figure 8: Selection of baud rate 500 kbit/s

After configuration of the CANopen Slave option module as variable "us_CoSl_Baudrate" in the global variable worksheet "bmaXX_Variables", the baud rate is available in the PROPROG wt II project.

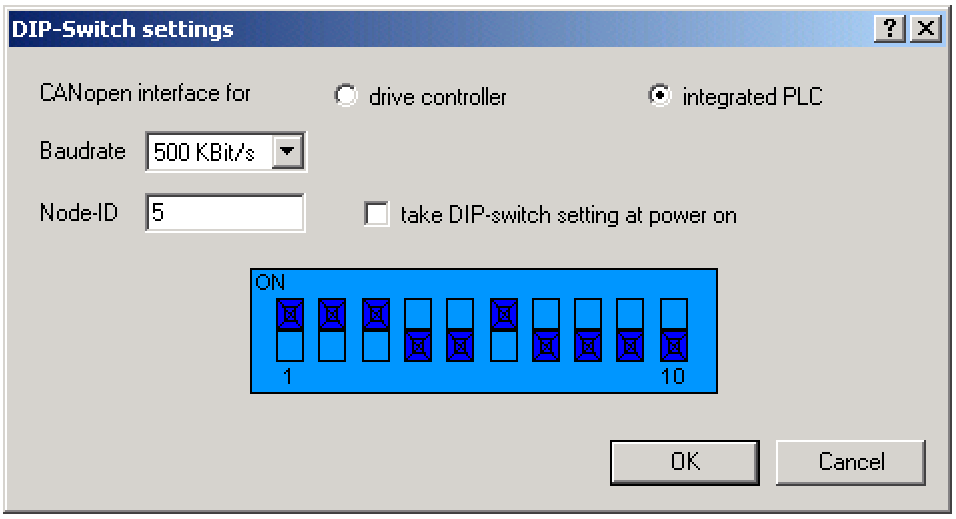
Continue with "Setting the Node ID".

3.3.6.2 Setting the Node ID

Set the Node ID (network node address, CANopen Slave address) with which the CANopen Slave option module is to report on the CANopen bus system as follows:

- Click on the "Settings" button in the "CANopen interface for" area in the CANopen Slave Configurator, provided that the "DIP-Switch settings" window has not yet opened.
- Click in the "Node ID" edit box in the "DIP-Switch settings" window.
- Enter the desired Node ID. You can assign a Node ID from 1 to 127 for the CANopen Slave option module. Other values are not permitted and will result in an error message

In the example, the Node ID "5" is set.


DIP-Switch settings

CANopen interface for [drive controller] [✓ integrated PLC]

Baudrate 500 KBit/s

Node-ID 5

[ ] take DIP-switch setting at power on

Figure 9: Setting the Node ID "5"

By entering a Node-ID, the display of DIP switches 4 to 10 is changed.

Please note that

```
        Node ID = "DIP switches 4 to 10" + 1.
```

I. e. with Node-ID "5", a "4" (DIP switch 6 = "ON") is displayed as the value of "DIP switches 4 to 10".

To set the Node-ID, you do not need to set DIP switches 4 to 10 on the CANopen Slave option module (the Node ID is set with FB CANop405_INIT_SL in the PROPROG wt II project).

Alternatively, you can also activate the "take DIP-switch setting at power on" check-box. In this case, the CANopen Slave option module accepts the Node ID of DIP switches 4 to 10.

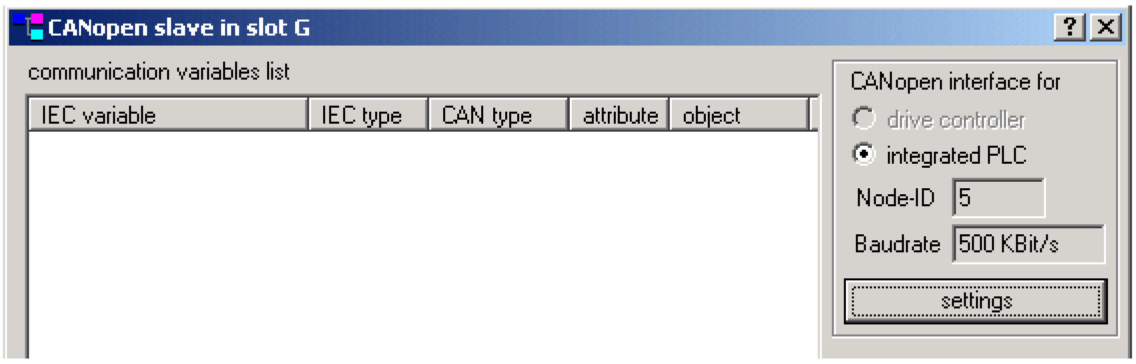
Please note that

```
        Node ID = "DIP switches 4 to 10" + 1.
```

I. e. with "DIP switches 4 to 10" = "4" (DIP switch 6 = "ON") the Node-ID is the value "5".

After configuration of the CANopen Slave option module as variable "us_CoSl_G_NodeID" in the global variable worksheet "bmaXX_Variables" the Node-ID is available in the PROPROG wt II project.

This completes the settings in the CANopen Slave Configurator in the range "CANopen interface for".


CANopen slave in slot G

communication variables list

IEC variable	IEC type	CAN type	attribute	object

CANopen interface for

( ) drive controller

(•) integrated PLC

Node-ID 5

Baudrate 500 KBit/s

Figure 10: Settings in the CANopen connection area

3.3.6.3 Creating variables

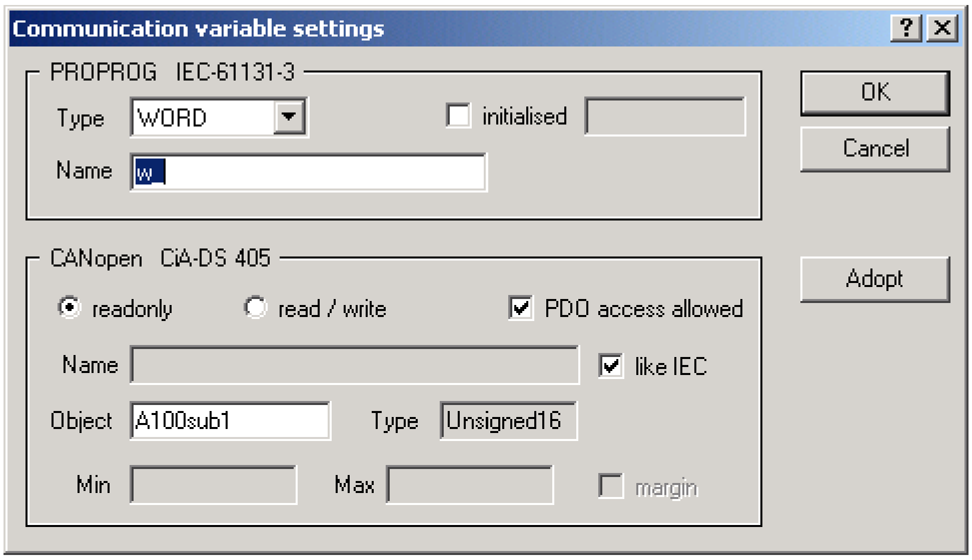
This chapter explains how to create an IEC 61131-3 variable and assign it to a CANopen object. This variable is then available in the PROPROG wt II project, in the global variables worksheet "bmaXX_Variables", for programming your application. In addition, the CANopen object (which is assigned to the variables) is available for communication via CANopen.

- ro (read only):
  The b maXX PLC transmits the data from the variables to the CANopen Slave option module, which then sends the data to the CANopen Master.
  From the viewpoint of the CANopen Master, these data are the actual values from the b maXX, i.e. data that are read.
- rw (read and write)
  The CANopen Master sends the data to the CANopen Slave option module, which then transmits the data to the b maXX PLC. The data are available in the variables on the b maXX PLC.
  From the viewpoint of the CANopen Master, these data are the reference values for the b maXX, i.e. data that are written.

To create a variable, click on the "Add" button in the CANopen Slave Configurator.

➲ Click on the "Add" button

The "Communication variable settings" window opens.



Communication variable settings

PROPROG IEC-61131-3

Type: WORD

Name: w

☐ initialised

CANopen CiA-DS 405

● readonly ○ read / write ☑ PDO access allowed

Name: ☑ like IEC

Object: A100sub1 Type: Unsigned16

Min: Max: ☐ margin

Figure 11: Opening of the "Communication variable settings" window

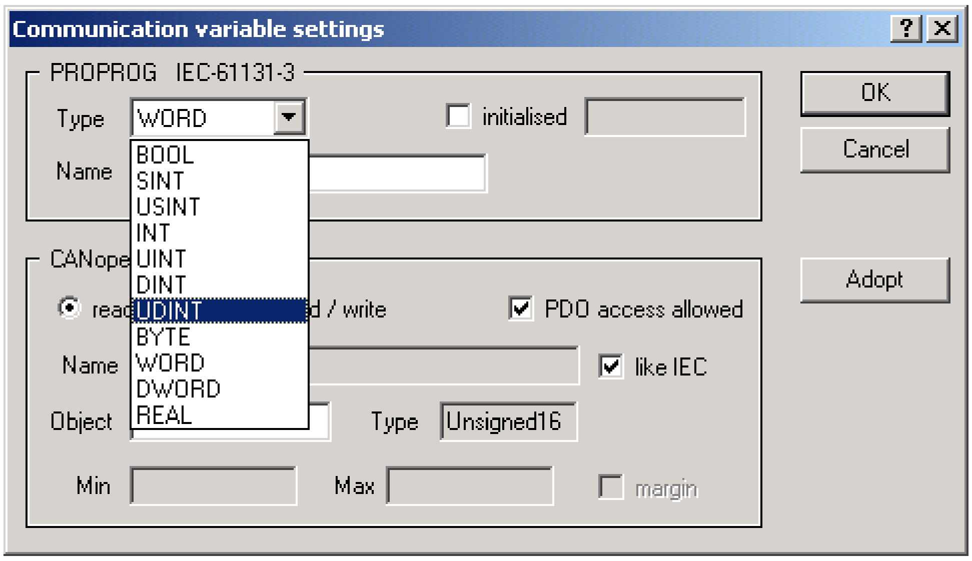
The window is organized into the areas "PROPROG IEC 61131-3" and "CANopen CiA DS 405".

Area PROPROG IEC 61131-3

Select the data type for the variables in the "Type" edit box in the PROPROG IEC 61131-3 area. The IEC 61131-3 data types BOOL, SINT, USINT, INT, UINT, DINT, UDINT, BYTE, WORD, DWORD and REAL are available for selection. For our example, please select the data type UDINT.

- Click in the "Type" edit box
- Select the desired data type from the list.

In the example, the data type UDINT is selected.



Communication variable settings

PROPROG IEC-61131-3

Type: WORD

BOOL, SINT, USINT, INT, UINT, DINT, UDINT, BYTE, WORD, DWORD, REAL

☐ initialised

CANopen

● readonly ○ read / write ☑ PDO access allowed

Name ☑ like IEC

Object Type: Unsigned16

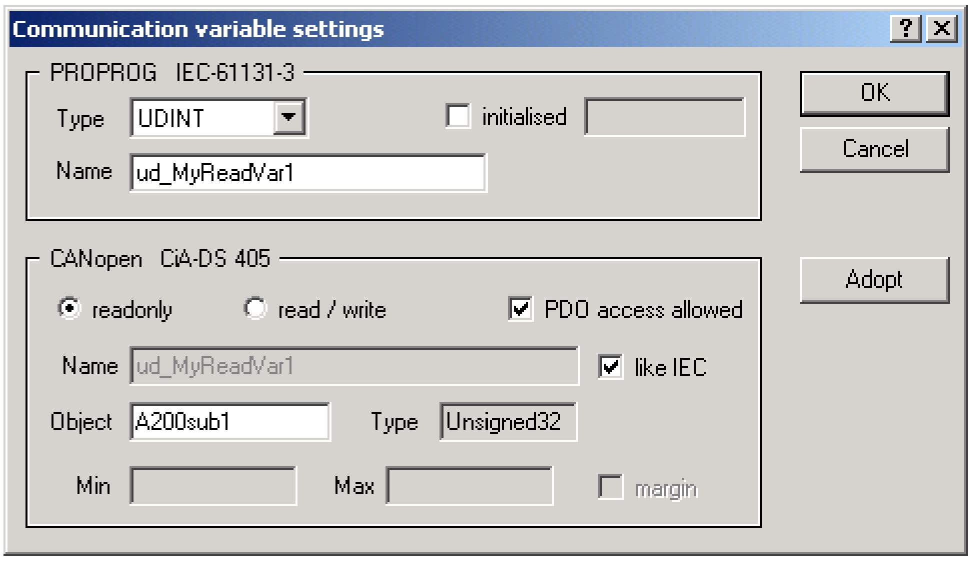
Min: Max: ☐ margin

Figure 12: Selecting the data type

The relevant data type short designation for the variable then appears in the "Name" edit box. You can now enter the variable name in the "Name" edit box (after the data type short designation).

- Enter the desired variable name in the "Name" edit box.

In the example, the variable name "ud_MyReadVar1" is entered.



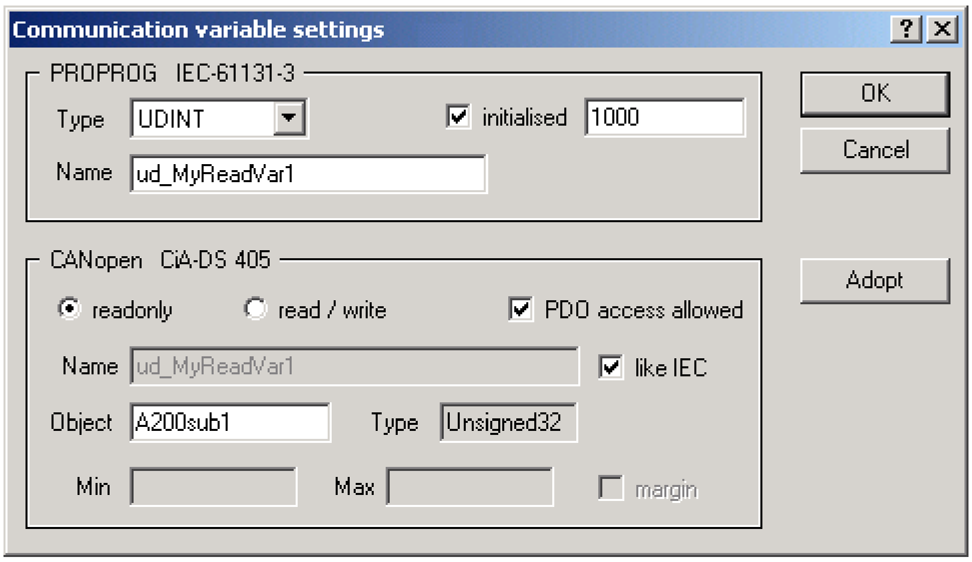
PROPROG IEC-61131-3
Type: UDINT
[ ] initialised:
Name: ud_MyReadVar1

CANopen CiA-DS 405
[✓ readonly] read / write
[✓] PDO access allowed
Name: ud_MyReadVar1
[✓] like IEC
Object: A200sub1
Type: Unsigned32
Min:
Max:
[ ] margin

Figure 13: Assigning the variable name

You can assign a start value for your variable in the "initialised" edit box.

- Click on the checkbox (on the left of the "initialised" edit box).
- Enter the start value for the variable in the "initialised" edit box.



PROPROG IEC-61131-3
Type: UDINT
[✓] initialised: 1000
Name: ud_MyReadVar1

CANopen CiA-DS 405
[✓ readonly] read / write
[✓] PDO access allowed
Name: ud_MyReadVar1
[✓] like IEC
Object: A200sub1
Type: Unsigned32
Min:
Max:
[ ] margin

Figure 14: Presetting the initialization value

This completes creation of the variables in the area "PROPROG IEC 61131-3". The variables are now assigned to a CANopen object in the range "CANopen CiA DS 405".

Area CANopen CiA DS 405

In this range, a CANopen object is assigned to the variables from the "PROPROG IEC 61131-3" range. Therefore the term "Object" is used in this range instead of the term "Variable".

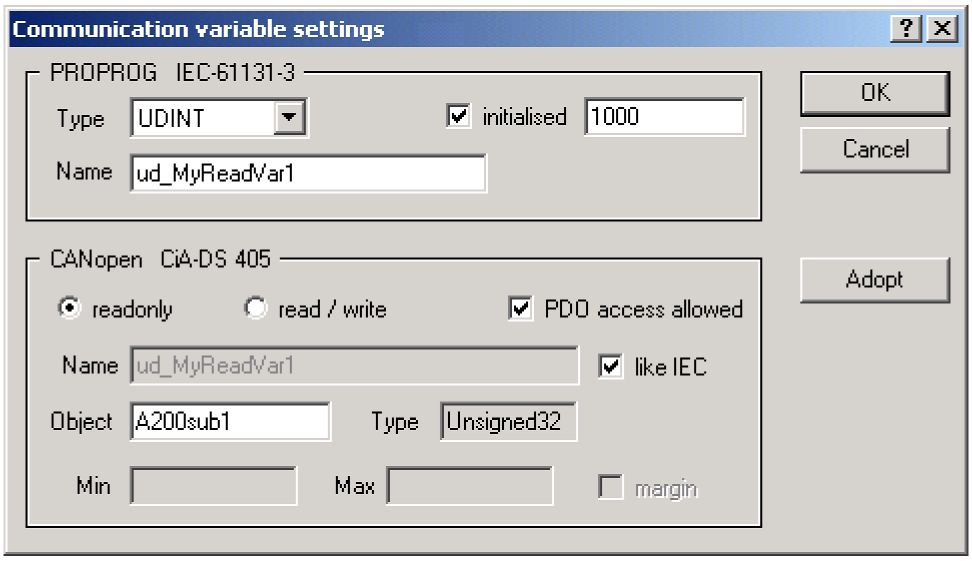
You can define the transfer direction of the object with the radio buttons "read only" and "read / write".

If you select "read / write", the object is received via the CANopen bus and written to the variable on the b maXX PLC. The object can only be assigned to an RxPDO of the CANopen Slave option module.

If you select "read only", the variable is read from the b maXX PLC and sent as object to the CANopen bus. The object can only be assigned to a TxPDO of the CANopen Slave option module.

With the "PDO access allowed" check box you can define whether this object can be used in PDOs, i. e. whether a mapping in PDOs is permitted for this object. This entry is necessary for the EDS file.

- Click on the "read only" radio button.
- Click on the "PDO access allowed" radio button.


Communication variable settings

PROPROG IEC-61131-3
Type: UDINT
[✓] initialised: 1000
Name: ud_MyReadVar1

CANopen CiA-DS 405
(●) readonly ( ) read / write
[✓] PDO access allowed
Name: ud_MyReadVar1
[✓] like IEC
Object: A200sub1
Type: Unsigned32
Min:
Max:
[ ] margin

Figure 15: PDO settings variable "ud_MyReadVar1"

You can give the object a symbolic name. For the sake of clarity, the symbolic name of the object is the same as the variable name (from the area "PROPROG IEC 61131-3"). If you wish to change the symbolic name of the object, deactivate the "like IEC" check box and then assign the new symbolic name for the object in the "Name" edit box. In the example, the symbolic name of the object has not been changed.

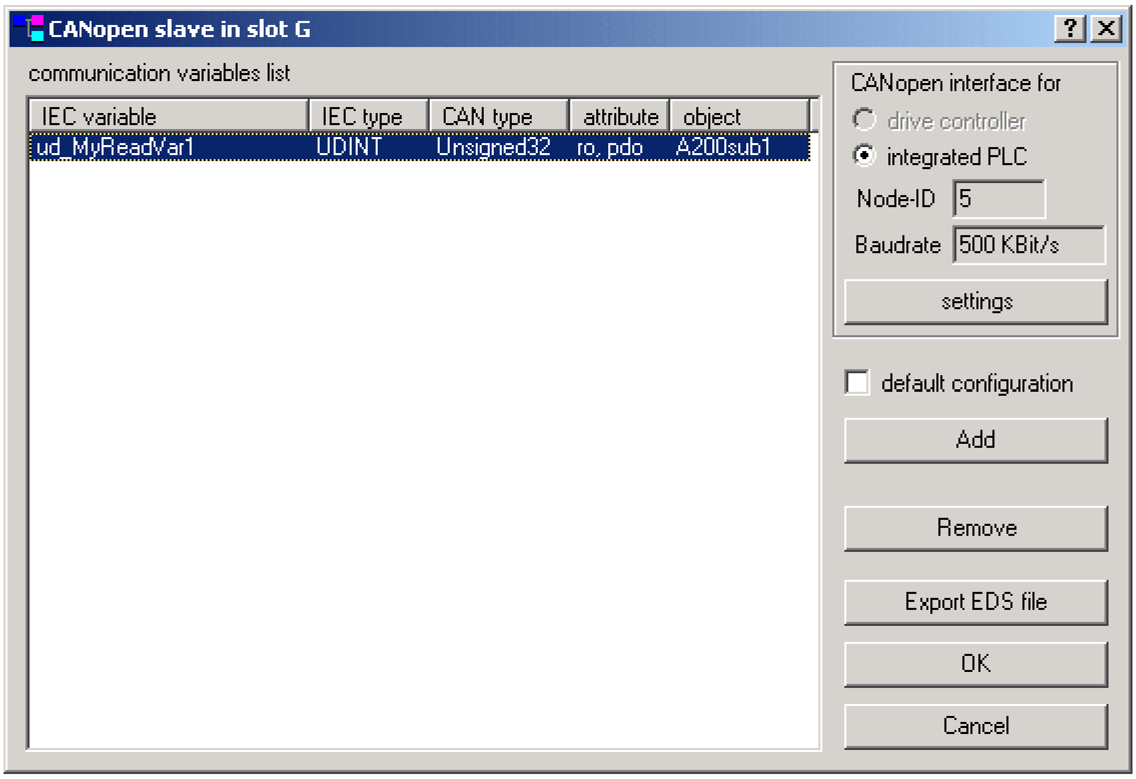
The object index and the object subindex of the (CANopen) object are entered in the "Object" edit box. The CANopen Slave Configurator automatically enters the next free object index and object subindex according to the CiA device profile DS 405 (see Appendix

▹B.2 Objects according to DS 405◃ from page 100 onward). You can also enter other object indices and/or object subindices according to DS 405. The system checks whether the object indices and object subindices are already being used, as well as checking whether the object indices are appropriate for the selected data type according to DS 405. In the example, the value suggested by the CANopen Slave Configurator, object index A200h and object subindex 1 (→ A200sub1), is not changed.

The "margin" checkbox and the "Min" and "Max" edit boxes are not used for "read only" objects.

This completes the assignment of variables from the area "PROPROG IEC 61131-3" to a CANopen object in the area "CANopen CiA DS 405". Click on the "OK" button

The "Communication variable settings" window is closed and the new variable is displayed in the "CANopen Slave in slot x" window (x = G to M).


CANopen slave in slot G

communication variables list

IEC variable	IEC type	CAN type	attribute	object
ud_MyReadVar1	UDINT	Unsigned32	ro, pdo	A200sub1

CANopen interface for
( ) drive controller
(●) integrated PLC
Node-ID: 5
Baudrate: 500 KBit/s

[ ] default configuration

Figure 16: Variable "ud_MyReadVar1"

We will now create three further variables and assign them to objects.

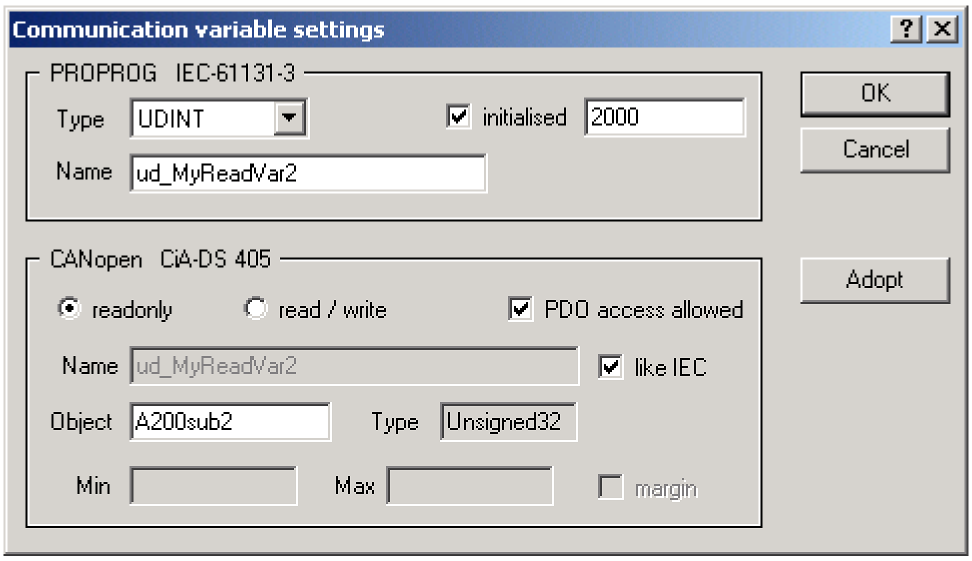
The variable "ud_MyReadVar2" is assigned to object index A200h and object subindex 2.

The variable "ud_MyWriteVar1" is assigned to object index A680h and object subindex 1.

The variable "u_MyWriteVar2" is assigned to object index A580h and object subindex 1.

- Click on the "Add" button
- Click in the "Type" edit box and select the data type UDINT from the list.
- Enter the variable name "ud_MyReadVar2" in the "Name" edit box.
- Click on the checkbox (to the left of the "initialised" edit box) and enter the start value "2000" for the variable in the "initialised" edit box.

- Click on the "read only" radio button.
- Click on the "PDO access allowed" checkbox.
- Enter "A200Sub2" in the "Object" edit box (this value should already have been entered by the CANopen Slave Configurator).

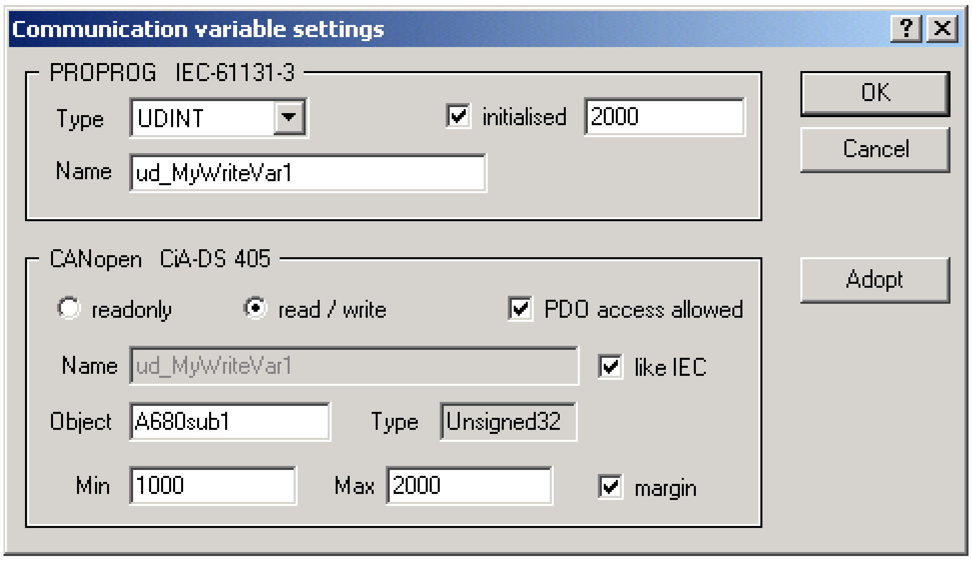


Type: UDINT
[✓] initialised: 2000
Name: ud_MyReadVar2

[✓ readonly] read / write
[✓] PDO access allowed
Name: ud_MyReadVar2
[✓] like IEC
Object: A200sub2
Type: Unsigned32
Min:
Max:
[ ] margin

Figure 17: PDO settings variable "ud_MyReadVar2"

- Click on the "Adopt" button.
- Select the data type UDINT from the list in the "Type" edit box (this value should already have been entered by the CANopen Slave Configurator).
- Enter the variable name "ud_MyWriteVar1" in the "Name" edit box.
- Click on the checkbox (to the left of the "initialised" edit box) and enter the start value "1500" for the variable in the "initialised" edit box.
- Click on the "read / write" radio button.
- Click on the "PDO access allowed" checkbox.
- Enter "A680Sub1" in the "Object" edit box (this value should already have been entered by the CANopen Slave Configurator).
- Click on the "margin" checkbox and enter the value 1000 in the "Min" edit box and the value 2000 in the "Max" edit box.
  These values are entered in the EDS file.
  With this, a CANopen Master can check whether the values that it transmits are within this range.
  In the CANopen Slave no check of this limitation takes place on the CANopen Slave option module. The user must check the values in the application program on the b maXX PLC of the CANopen Slave (here CANopen Slave means the b maXX 4400 device with b maXX PLC and CANopen Slave option module).


Communication variable settings

PROPROG IEC-61131-3

Type: UDINT — [✓] initialised: 2000

Name: ud_MyWriteVar1

CANopen CiA-DS 405

( ) readonly [● read / write] — [✓] PDO access allowed

Name: ud_MyWriteVar1 — [✓] like IEC

Object: A680sub1 — Type: Unsigned32

Min: 1000 — Max: 2000 — [✓] margin

Figure 18: PDO settings variable "ud_MyWriteVar1"

- Click on the "Adopt" button.
- Select the data type UINT from the list in the "Type" edit box.
- Enter the variable name "u_MyWriteVar2" in the "Name" edit box.
- This variable is not given a start value.
- Click on the "read / write" radio button.
- Click on the "PDO access allowed" checkbox.
- Enter "A580Sub1" in the "Object" edit box (this value should already have been entered by the CANopen Slave Configurator).

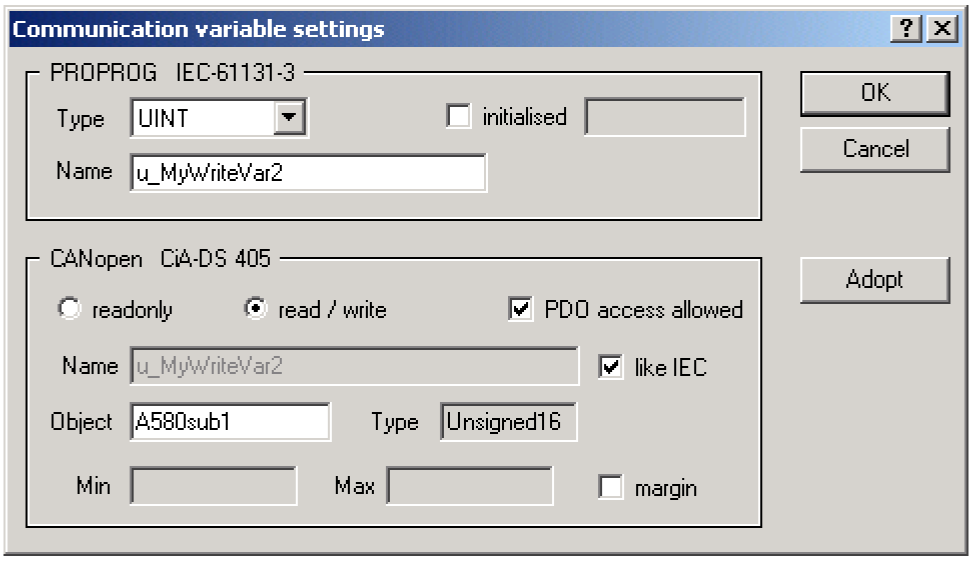
No limitation is entered for this object.

The CANopen Slave configurator uses the upper and lower limit of the data type value range as upper and lower limit (in this case 0x0000 and 0xFFFF for data type UINT).

These values are entered in the EDS file.

With this, a CANopen Master can check whether the values that it transmits are within this range.

In the CANopen Slave no check of this limitation takes place on the CANopen Slave option module. The user must check the values in the application program on the b maXX PLC of the CANopen Slave (here CANopen Slave means the b maXX 4400 device with b maXX PLC and CANopen Slave option module).



Communication variable settings

PROPROG IEC-61131-3

Type: UINT

[ ] initialised

Name: u_MyWriteVar2

CANopen CiA-DS 405

( ) readonly [✓ read / write]

[✓ PDO access allowed]

Name: u_MyWriteVar2

[✓ like IEC]

Object: A580sub1

Type: Unsigned16

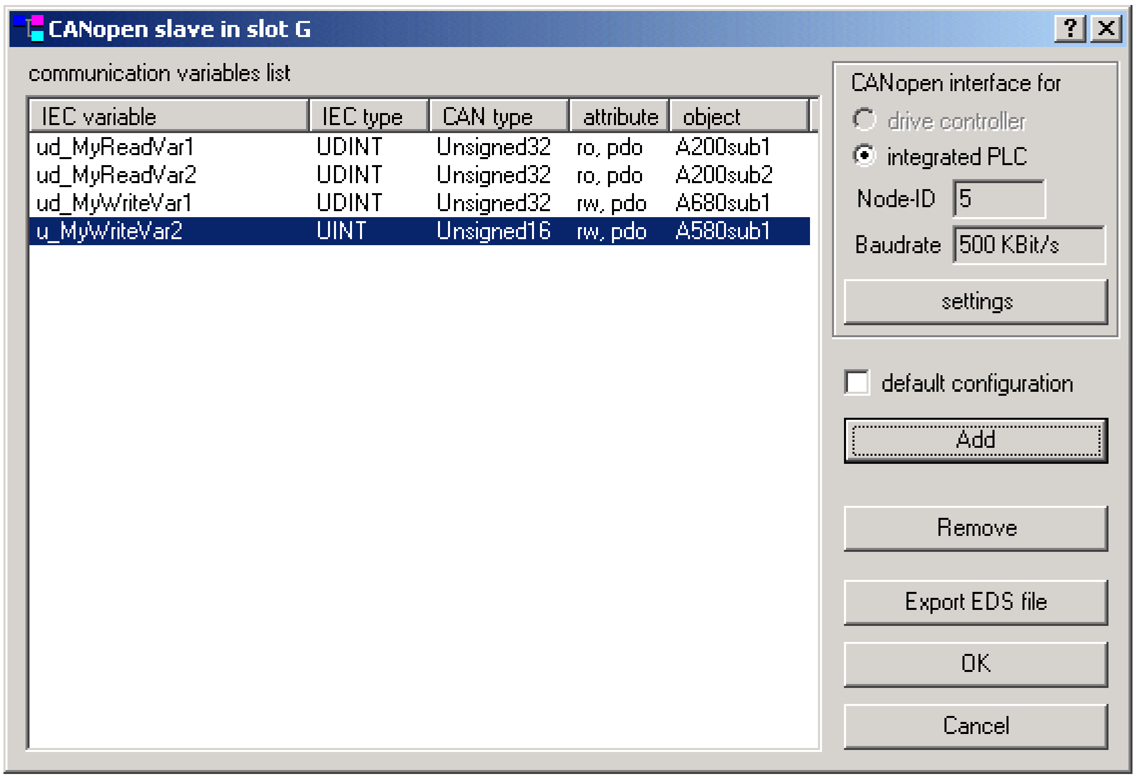
Min:

Max:

[ ] margin

Figure 19: PDO settings variable "u_MyWriteVar2"

- Click on the "OK" button". The "Communication variable settings" window is closed and the new variables are displayed in the "CANopen Slave in slot x" window (x = G to M).



CANopen slave in slot G

communication variables list

IEC variable	IEC type	CAN type	attribute	object
ud_MyReadVar1	UDINT	Unsigned32	ro, pdo	A200sub1
ud_MyReadVar2	UDINT	Unsigned32	ro, pdo	A200sub2
ud_MyWriteVar1	UDINT	Unsigned32	rw, pdo	A680sub1
u_MyWriteVar2	UINT	Unsigned16	rw, pdo	A580sub1

CANopen interface for

( ) drive controller [✓ integrated PLC]

Node-ID: 5

Baudrate: 500 KBit/s

[ ] default configuration

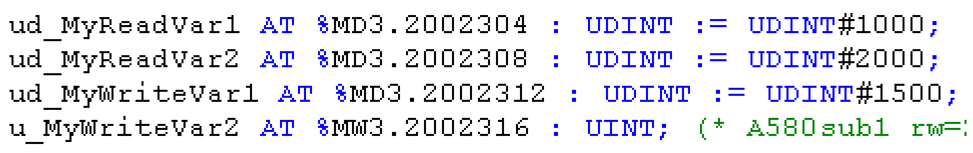
Figure 20: Created variables

This completes creation of the variables "ud_MyReadVar1", "ud_MyReadVar2", "ud_MyWriteVar1" and "u_MyWriteVar1" and their assignment to the objects "A200sub1", "A200sub2", "A680sub1" and "A580sub1".

To generate an EDS file, please read section ▹Generating an EDS file◃ on page 42.

- End the CANopen Slave Configurator by clicking on the "OK" button.
- End the b maXX Configurator with File\Exit and confirm the "...Save?" query with "Yes".

The global variable worksheet "bmaXX_Variables" is now generated for the PROPROG wt II project and inserted or updated in the project tree. The variables for programming the application are available in the PROPROG wt II project.



```
ud_MyReadVar1 AT %MD3.2002304 : UDINT := UDINT#1000;
ud_MyReadVar2 AT %MD3.2002308 : UDINT := UDINT#2000;
ud_MyWriteVar1 AT %MD3.2002312 : UDINT := UDINT#1500;
u_MyWriteVar2 AT %MW3.2002316 : UINT; (* A580sub1 rw=
```

Figure 21: Global variable worksheet "bmaXX_Variables"

3.3.6.4 Generating an EDS file

The EDS file (EDS - Electronic Data Sheet) describes a CANopen device on the CANopen bus. Amongst other things, this contains the information required for communication with the CANopen device. You need this EDS file to make the CANopen device "b maXX 4400 with b maXX PLC and CANopen Slave" option module usable in a CANopen Master configurator. The name extension of the EDS file is "eds" (e. g. MyEDS-File.eds)

NOTE

The CANopen Slave (device b maXX 4400 with b maXX PLC and CANopen Slave option module) is IEC 61131-3-programmable, thanks to the b maXX PLC.

With these devices, the contents of the EDS file change as soon as

- other variables and/or object assignments are used than those defined in the default setting
- other variables and/or object assignments are used than those defined in the last setting

A change to the default setting or the last setting for the variables and their assignment to the objects inevitably entails generation of a new EDS file (and integration of this EDS file into the CANopen Master Configurator).

After creating the variables in the PROPROG wt II project in the section "Creating variables" and assigning them to CANopen objects, the EDS file can be generated.

If you have not assigned any variables and objects, you can also press the default button. The default assignment of variables and objects is then used.

NOTE

Clicking on the "Default configuration" checkbox will delete any existing variables in the CANopen Slave configurator!

NOTE

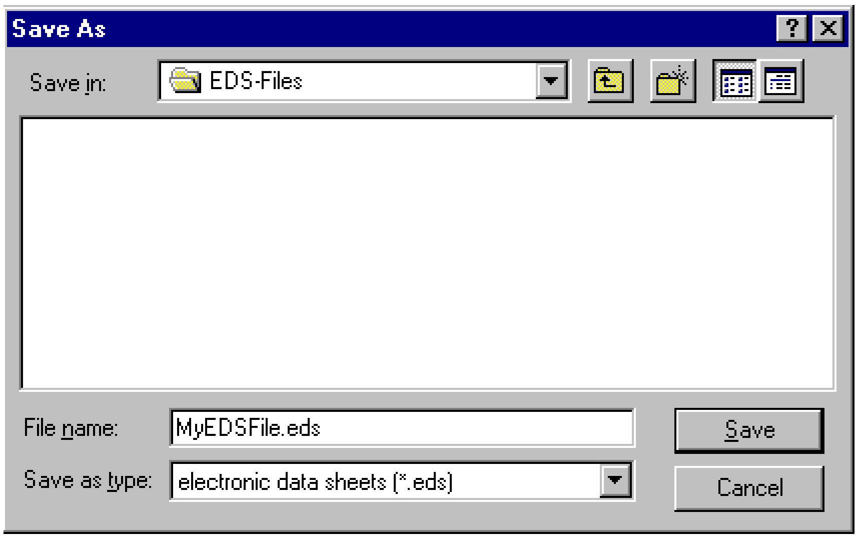
The default EDS file for the CANopen Slave option module for b maXX PLC is located on the CD "CANop405_PLC01_20bd03" (or higher), which also contains the library "CANop405_PLC01_20bd03" (or higher).

In order to generate the EDS file, you must open the "CANopen Slave in slot G" window (PROPROG wt II project → Configuration_Tools → Drive_Configuration → Option modules \ G:CAN-03-00-00 → context menu item "Configure").

To generate the EDS file, click on the "Export EDS file" button and enter a file name and the storage location for the EDS file.

- Open the CANopen Slave Configurator, specify the desired variables and assign CANopen objects to them (see ▹Creating variables ◃ on page 34).
- Click on the "Export EDS file" button. The "Save As" window opens.
- Find the storage location in the "Save As" window, e. g. your self-generated folder EDS-Files, and enter the name of the EDS file in the "File name" edit box.

In the example, the EDS file is called "MyEDSFile.eds"



File name: MyEDSFile.eds

Save as type: electronic data sheets (*.eds)

Figure 22: Export EDS file

- Store the file by clicking on the "Save" button.
- End the CANopen Slave Configurator by clicking on the "OK" button. Close the b maXX Configurator with File\End or by clicking on [x].

The EDS file is available for use in CANopen Master configurators.

3.3.7 Programming the application in the PROPROG wt II project

3.3.7.1 Initialization of the CANopen Slave option module

When the CANopen Slave option module is initialized, the initialization data are transferred to the option module. These initialization data include baud rate and Node ID.

Procedure for initializing the CANopen Slave option module

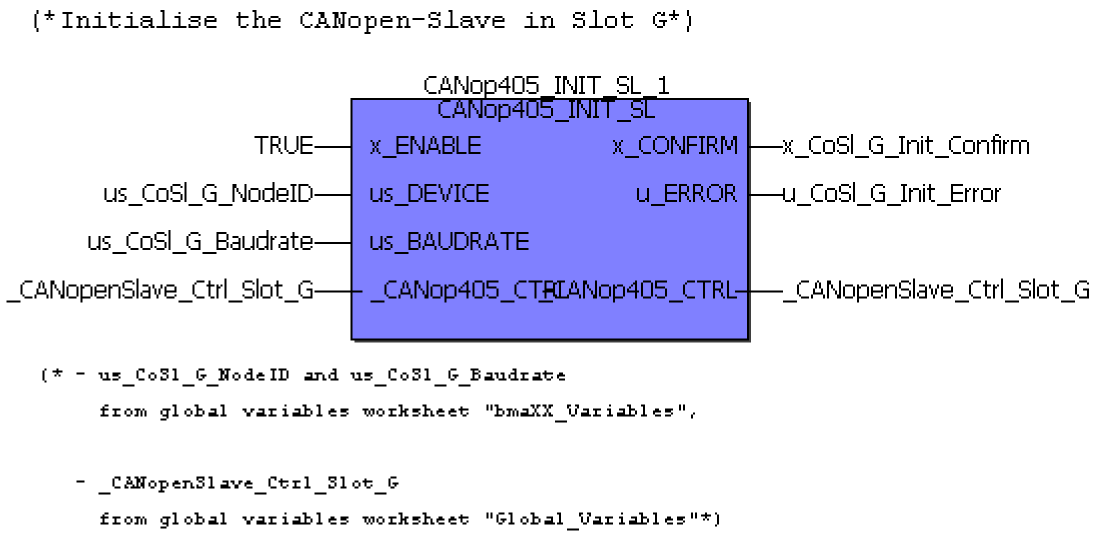
The CANopen Slave option module is initialized by the FB CANop405_INIT_SL. To use this function block, follow the steps below:

- Create a POU with PLC type SH03_30 and processor type BM4_O_PLC01. This POU is to be called up subsequently in a cold start / warm start task.
- Place the FB CANop405_INIT_SL in this POU.
- Interconnect the function block with variables of the correct data type.
- At the us_DEVICE input, connect the global variable "us_CoSl_G_NodeID" from the "bmaXX_Variables" global variable worksheet. This variable was created by the CANopen Slave Configurator, likewise the value of this variable (see section ▹Setting the Node ID ◃ on page 32).
- At the us_BAUDRATE input, connect the global variable "us_CoSl_G_Baudrate" from the "bmaXX_Variables" global variable worksheet. This variable was created by the CANopen Slave Configurator, likewise the value of this variable (see section ▹Setting the baud rate ◃ on page 32). The meaning of the value of this variable is indicated in the following table:

us_Baudrate	Network baud rate
0	**1 Mbps**
1	Reserved
2	**500 kbps**
3	**250 kbps**
4	**125 kbps**
5	Reserved
6	Reserved
7	**20 kbps**
8 - 255	Reserved

Example: Initialization of the CANopen Slave option module

The interconnection of the FB CANop405_INIT_SL has the following construction:


```
(*Initialise the CANopen-Slave in Slot G*)
```

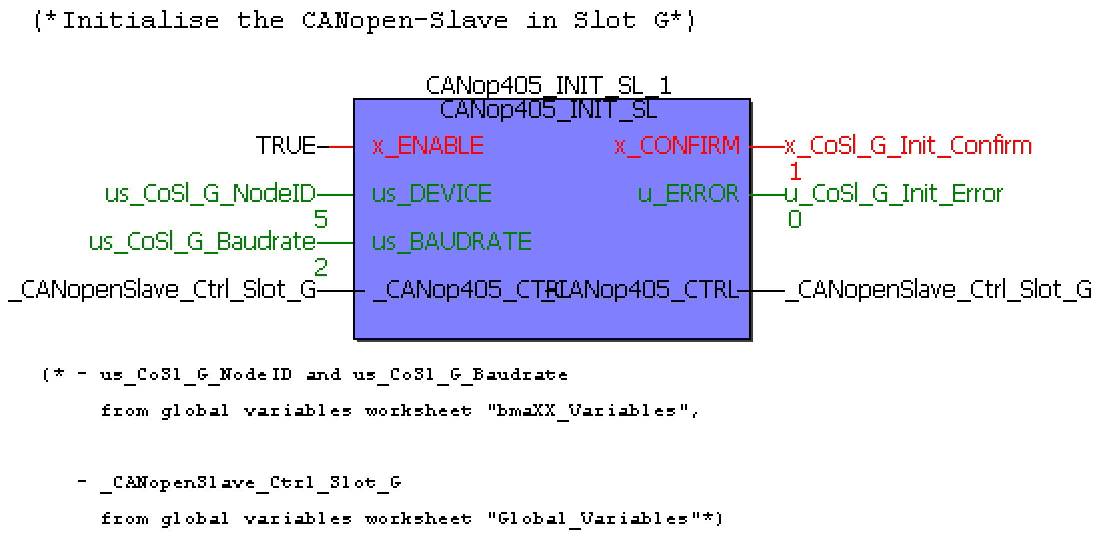
```
(* - us_CoSl_G_NodeID and us_CoSl_G_Baudrate
     from global variables worksheet "bmaXX_Variables",

   - _CANopenSlave_Ctrl_Slot_G
     from global variables worksheet "Global_Variables"*)
```

Figure 23: Initialization of the CANopen Slave with the FB CANop405_INIT_SL

- Create a task for cold start and warm start of the PLC if these are not yet available in your project. Link the created POU for initialization of the option module in the two tasks.
- Compile the project and send it to the PLC as boot project. Switch the b maXX 4400 device off and on again.

The FB CANop405_INIT_SL reports successful initialization with x_CONFIRM = 1 and u_ERROR = 0.

Example:


```
(*Initialise the CANopen-Slave in Slot G*)
```

```
(* - us_CoSl_G_NodeID and us_CoSl_G_Baudrate
     from global variables worksheet "bmaXX_Variables",

   - _CANopenSlave_Ctrl_Slot_G
     from global variables worksheet "Global_Variables"*)
```

Figure 24: Initialization of the CANopen Slave with the FB CANop405_INIT_SL - Online

Successful initialization is also indicated by the LEDs at the sockets of the option module:

H1 flashing *) H2 off H3 off H4 off	CANopen: The CANopen Slave option module is in PRE-OPERATIONAL NMT state and ready for data transfer using SDOs.
H1 off or flashing *) H2 flashing *) H3 off H4 off	Network: The CANopen Slave option module has detected an error in the network. Check whether the communication cable is correctly connected.

*) flashing: on approx. 200ms, then off approx. 200 ms

Set the correct baud rate on the remaining CANopen participants. Refer to the relevant documentation.

3.3.7.2 Network commands - NMT

Definition according to CANopen specification

The communication states of network nodes are controlled through a CANopen Master using NMT- (Network Management) message frames. The device has the following communication states:

- INITIALIZATION
- PRE-OPERATIONAL
- STOPPED
- OPERATIONAL

The behavior of individual network nodes is described by the following state transitions:

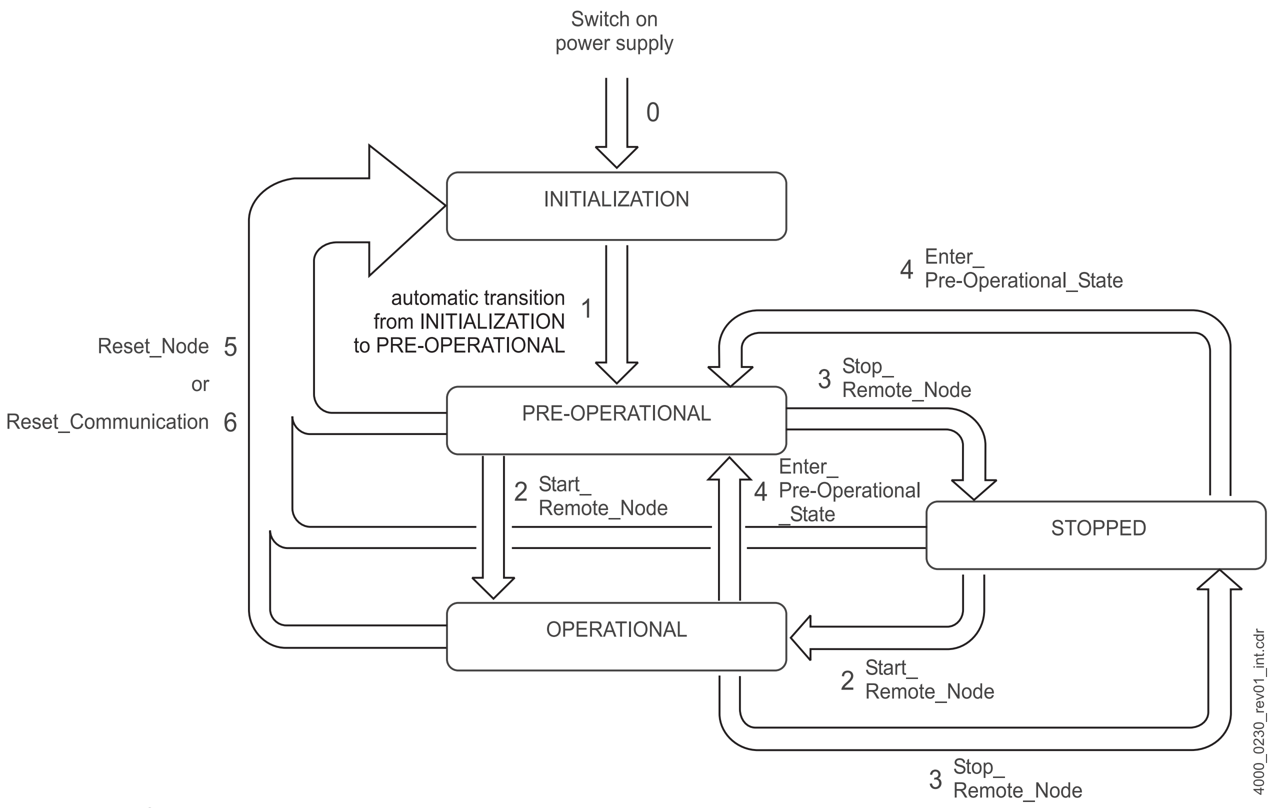


Figure 25: State transitions

Directly after activation, a CANopen Slave is in INITIALIZATION state and enters the PRE-OPERATIONAL state automatically.

Before the CANopen Slave enters PRE-OPERATIONAL, it sends its Boot Up message frame. With the Boot Up message frame, the CANopen Slave reports its readiness for configuration via SDOs on the CAN bus. The user has no influence on sending the boot up message frame.

In PRE-OPERATIONAL, a CANopen Slave can be configured via SDOs. Data exchange via PDOs is not possible.

In STOPPED, Node Guarding and Network Management are possible. Neither SDOs nor PDOs can be transmitted or received.

When the CANopen Slave is in OPERATIONAL, SDOs, PDOs, NMT, EMCY and Node Guarding or heartbeat message frames can be sent and received.

The individual state transitions are initiated by an NMT master. Only the change from INITIALIZATION to PRE-OPERATIONAL occurs automatically in the CANopen Slave and does not require an NMT command.

The CANopen Slave option module can process the following NMT commands (except **1**):

1 Automatic transition from INITIALIZATION to PRE-OPERATIONAL

2 Start_Remote_Node

3 Stop_Remote_Node

4 Enter_Pre-Operational_State

5 Reset_Node

6 Reset_Communication

To start the CANopen Slave using NMT commands, please refer to the documentation for the NMT Master. Generally, the CANopen Master is also the NMT Master.

The CANopen Slave changes its own state according to the state transitions sent (by the CANopen Master) via NMT commands.

A CANopen Slave can report the following states:

- PRE-OPERATIONAL
- STOPPED
- OPERATIONAL

The states are assigned the following numbers:

Status	**Number**
STOPPED	4
OPERATIONAL	5
PRE-OPERATIONAL	127

Determine NMT state

You can determine the NMT state of the CANopen Slave option module in the application program on the b maXX PLC using the FB CANop405_GET_STATE_SL.

To do this, please follow the steps below

- Create a POU with PLC type SH03_30 and processor type BM4_O_PLC01. This POU is to be subsequently called up in a cyclical task.
- Place the FB CANop405_GET_STATE_SL in this POU.
- Interconnect the function block with variables of the correct data type.
- Create a cyclical task with average to low priority, if this is not already present in your project. Link the created POU (with the FB CANop405_GET_STATE_SL) in this task.
- Compile the project and send it to the PLC as (boot) project.
- Start the project.

The NMT state of the CANopen Slave option module is read with x_ENABLE = 1. The FB CANop405_GET_STATE_SL reports successful execution with x_CONFIRM = 1. In addition, the FB outputs the NMT state number at the us_STATE output and displays the NMT state at the s_STATE output.

Example: Determine NMT state

In order to be able to use the FB CANop405_GET_STATE_SL, we create a POU and place the function block in it. We assign the generated POU a cyclical task with 200 ms call interval. After interconnecting the function block, compile the project and send it as project to the b maXX PLC.

Start the project. The NMT state of the CANopen Slave (OPERATIONAL in our case) is read with x_ENABLE = 1:

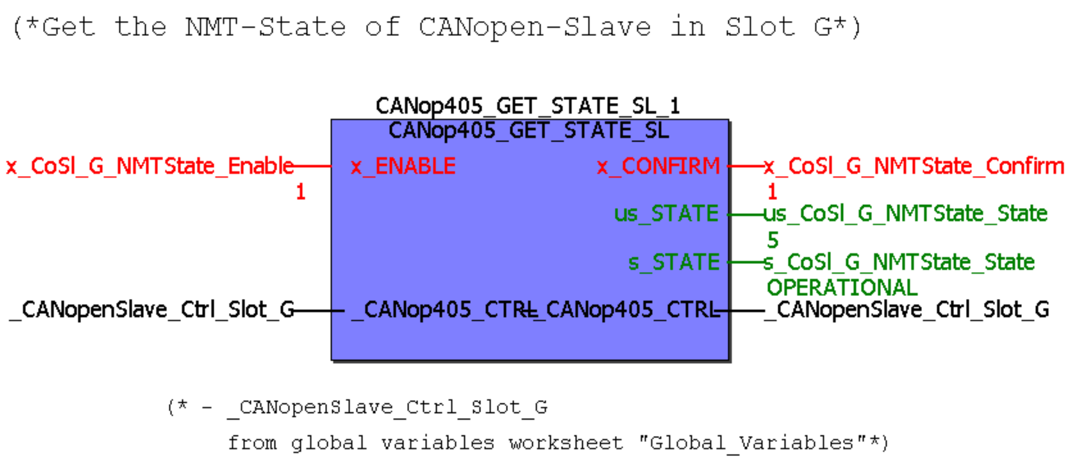

(*Get the NMT-State of CANopen-Slave in Slot G*)
CANop405_GET_STATE_SL_1
CANop405_GET_STATE_SL
x_CoSl_G_NMTState_Enable
1
x_ENABLE
x_CONFIRM
x_CoSl_G_NMTState_Confirm
1
us_STATE
us_CoSl_G_NMTState_State
5
s_STATE
s_CoSl_G_NMTState_State
OPERATIONAL
_CANopenSlave_Ctrl_Slot_G
_CANop405_CTRL
_CANop405_CTRL
_CANopenSlave_Ctrl_Slot_G
(* - _CANopenSlave_Ctrl_Slot_G
from global variables worksheet "Global_Variables"*)

Figure 26: NMT state of the CANopen Slave

3.3.7.3 Emergency message frames

Definition according to CANopen specification

The emergency message frames are used for error reporting by the network nodes (CANopen Slaves). The transfer of the message frame occurs according to the consumer/producer communication model, i. e. a network node (CANopen Slave) generates an emergency message frame, which is processed by one or several other network nodes. There is no confirmation of the receipt of the message frame. With each newly added error, a new emergency message frame will be transmitted once. Even if an error ceases to exist, a new emergency message frame is sent. The message frame is not repeated.

The CANopen Slave option module can send emergency message frames to other network nodes. The evaluation of emergency message frames from other network nodes is not possible.

The user data area of the emergency message frame is organized into 3 sections:

1 Emergency Error Code, 2 byte

2 Error Register, 1 byte

3 Manufacturer-specific Error Code, 5 byte

The Emergency Error Code has the following meaning according to CiA:

Error Code	Meaning
00xxh	Error Reset or No Error
10xxh	Generic error
20xxh	Current
21xxh	Current, device input side
22xxh	Current inside the device
23xxh	Current, device output side
30xxh	Voltage
31xxh	Mains Voltage

Error Code	Meaning
32xxh	Voltage inside the device
33xxh	Output Voltage
40xxh	Temperature
41xxh	Ambient temperature
42xxh	Device Temperature
50xxh	Device Hardware
60xxh	Device Software
61xxh	Internal Software
62xxh	User Software
63xxh	Data Set
70xxh	Additional Modules
80xxh	Monitoring
81xxh	Communication
8110h	CAN Overrun (Objects lost)
8120h	CAN in Error Passive Mode
8130h	Life Guard Error or Heartbeat Error
8140h	recovered from bus off
8150h	Transmit COB ID
82xxh	Protocol Error
8210h	PDO not processed due to length error
8220h	PDO length exceeded
90xxh	External Error
F0xxh	Additional Functions
FFxxh	Device-specific

xx = Not defined

Object 1001h is contained in the Error Register. A network node internal error can be entered in object 1001h. The coding according to CiA has the following construction:

Bit	Meaning
0	Generic error
1	Current
2	Voltage
3	Temperature
4	Communication error
5	Device profile-specific

Bit	Meaning
6	Reserved
7	Manufacturer-specific

5 bytes are available for a manufacturer-specific error code, for further characterization of errors.

Via the CANopen Slave option module on the b maXX PLC, you can send an emergency message frame with Emergency Error Code 0x6200, Error Register 0x80 [1] and any value for the manufacturer-specific error code.

Send emergency message frame

You can send an emergency message frame from the CANopen Slave option module in the application program on the b maXX PLC using the FB CANop405_EMCY_SEND_SL.

To do this, please follow the steps below

- Create a POU with PLC type SH03_30 and processor type BM4_O_PLC01. This POU is to be subsequently called up in a cyclical task.

Alternatively, you can also use the POU in which the NMT state is read.

- Place the FB CANop405_EMCY_SEND_SL in this POU.
- Interconnect the function block with variables of the correct data type. Enter any error code at the a_MAN_SPEC_ERR_CODE input. Access to the 5 bytes occurs via the entries a_MAN_SPEC_ERR_CODE[0] to a_MAN_SPEC_ERR_CODE[4].
- Create a cyclical task with average to low priority, if this is not already present in your project. Link the created POU (with the FB CANop405_GET_STATE_SL) in this task.

If you use the POU in which the NMT state is read, you have already assigned this POU to a task.

- Compile the project and send it to the PLC as (boot) project.
- Start the project.

The emergency message frame is sent by the CANopen Slave option module with x_ENABLE = 1. The FB CANop405_EMCY_SEND_SL reports successful execution with x_CONFIRM = 1.

Example: Send emergency message frame

In order to be able to use the FB CANop405_EMCY_SEND_SL, we create a POU and place the function block in it. We assign the generated POU a cyclical task with 200 ms call interval. After interconnecting the function block, compile the project and send it as project to the b maXX PLC. Start the project. The emergency message frame is sent by the CANopen Slave option module with x_ENABLE = 1:

[1] Value 0x80 in the Error Register is OR-linked to the entries in object 0x1001 on the CANopen Slave option module.

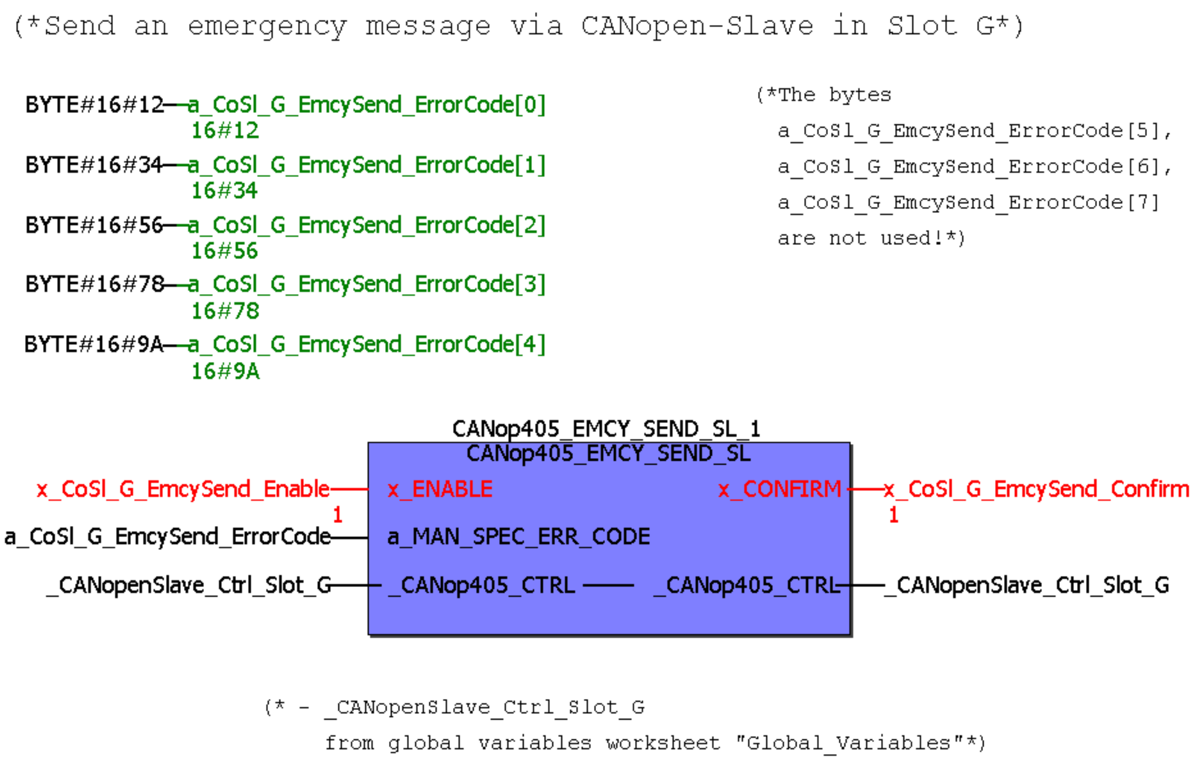

(*Send an emergency message via CANopen-Slave in Slot G*)
BYTE#16#12 a_CoSl_G_EmcySend_ErrorCode[0] 16#12
BYTE#16#34 a_CoSl_G_EmcySend_ErrorCode[1] 16#34
BYTE#16#56 a_CoSl_G_EmcySend_ErrorCode[2] 16#56
BYTE#16#78 a_CoSl_G_EmcySend_ErrorCode[3] 16#78
BYTE#16#9A a_CoSl_G_EmcySend_ErrorCode[4] 16#9A
(*The bytes a_CoSl_G_EmcySend_ErrorCode[5], a_CoSl_G_EmcySend_ErrorCode[6], a_CoSl_G_EmcySend_ErrorCode[7] are not used!*)
CANop405_EMCY_SEND_SL_1
CANop405_EMCY_SEND_SL
x_CoSl_G_EmcySend_Enable
1
x_ENABLE
x_CONFIRM
x_CoSl_G_EmcySend_Confirm
1
a_CoSl_G_EmcySend_ErrorCode
a_MAN_SPEC_ERR_CODE
_CANopenSlave_Ctrl_Slot_G
_CANop405_CTRL
_CANop405_CTRL
_CANopenSlave_Ctrl_Slot_G
(* - _CANopenSlave_Ctrl_Slot_G from global variables worksheet "Global_Variables"*)

Figure 27: Send emergency message frame

3.3.7.4 Service data exchange with SDOs

Definition according to CANopen specification

An SDO communication corresponds to the client/server communication model, i. e. the CANopen Master option module is the client and sends a message frame to a network node with the instruction to accept or send data. The network node acts as a server, accepts the data and confirms this with a message frame or sends the requested data.

The CANopen Slave option module is such a network node and consequently acts as data server. With the CANopen Slave option module, "expedited" SDO transfer according to DS 301 can be executed with up to 4 bytes of data per instruction. "Segmented" and "Block" SDO transfer according to DS 301 are not supported.

Writing and reading objects via SDO

The requirement data exchange is limited on the PROPROG wt II side to reading and writing global variables, which you have created under ▷Creating variables ◁ on page 34 with the CANopen Slave Configurator.

In the example, the variable "ud_MyReadVar2" was created in the CANopen Slave Configurator. The variable was also assigned to object index A200h and object subindex 2. A CANopen Master now has read-access to the object via the object index and subindex and can thus read the contents of the variable "ud_MyReadVar2". As the variable "ud_MyReadVar2" has been preconfigured with the start value 2000 (initial value), the value 2000 must be displayed in the CANopen Master when reading the variables.

In addition, the variable "u_MyWriteVar2" has been created in the CANopen Slave Configurator. The variable was also assigned to object index A580h and object subindex 1. A CANopen Master now has write-access to the object via the object index and subindex and can thus write to the contents of the variable "u_MyWriteVar2".

3.3.7.5 Process data exchange with PDOs

Definition according to CANopen specification

A PDO communication (PDO = Process Data Object) corresponds to the consumer/producer communication model, i. e. a network node generates a PDO message frame, which is processed by one or several other network nodes. There is no confirmation of the receipt of the message frame. The CANopen Slave option module can also generate and process PDO message frames. The transfer itself must be configured. You must therefore specifically define when the data in a PDO are to be transferred, and at which time the data in a PDO are to be entered. In contrast to data exchange via SDOs, with PDOs no direct access is made to an object in a network node. Rather, a network node is informed (during configuration by the CANopen Master) which objects are contained in a PDO message frame, with index, subindex and data length. These objects are then removed when writing a PDO and entered when reading a PDO. Up to 8 bytes of data are available per message frame.

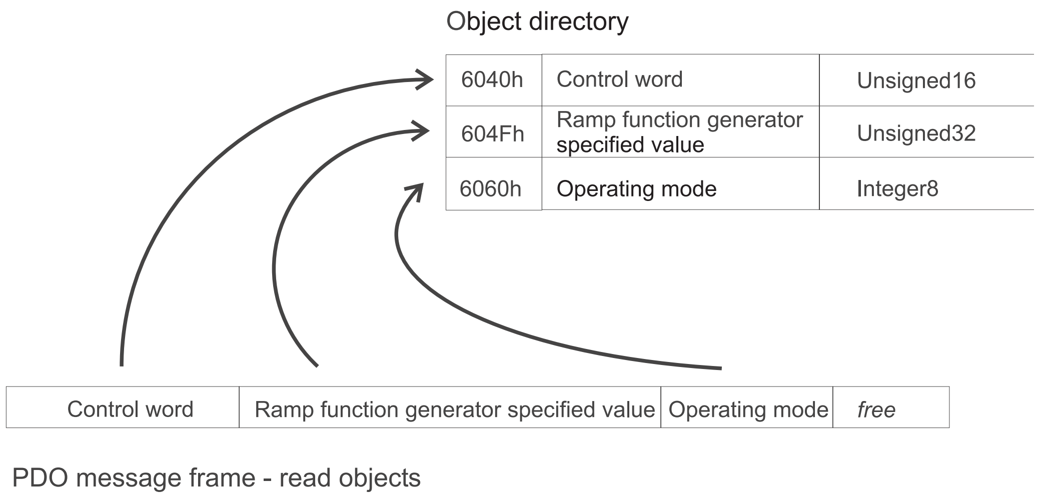


Figure 28: Reading objects from a PDO message frame

Only user data are contained in the PDO message frame itself; information on object index or subindex is no longer present, as this is already known to the network node due to the previously performed configuration. This process - mapping objects in a PDO - is also called "Mapping". A default mapping is already stored on the CANopen Slave option module, which can be modified and stored by the CANopen Master.

NOTE

The mapping is set by the CANopen Master if the default mapping is not to be used.

Data exchange via PDOs can be divided into three phases:

- Configuration of the transfer behavior of a PDO in each network node
- Configuration of the PDO mapping in each network node
- Cyclic data exchange via PDOs - sending and receiving objects via PDO

The configuration of the PDOs is stored in communication profile-specific objects of a device. These objects are read and written via SDOs (see Chapter ▹Service data exchange with SDOs ◃ on page 52). The configuration of PDOs may only occur in the PRE-OPERATIONAL of a network node.

After configuration of the transfer properties and mapping of the PDO, the actual data exchange via PDOs can begin. The CANopen Slave option module receives PDOs that are transmitted by the other network nodes and responds according to its settings and the application program on the b maXX PLC. Likewise, the CANopen Slave option module can transmit PDOs which are received from the other network nodes. In principle, data exchange via PDOs is only possible when the network nodes are in OPERATIONAL. No further configuration of the PDOs is possible in OPERATIONAL.

Please note that data exchange via PDOs must also always be considered from the view of the network nodes, i. e. when the CANopen Slave sends a PDO, this is a Send-PDO (Transmit-PDO, TxPDO) on both slave side and master side. When the CANopen Slave receives a PDO, this is a Receive-PDO on both slave side and on master side (Receive-PDO, RxPDO).

Configuration of the transfer properties of a PDO

The properties for transfer of a PDO are described by communication profile-specific objects of a network node. The configuration object for the first Receive-PDO (1400h) and the first Transmit-PDO (1800h) is described below. To configure all further PDOs, the next higher object index must be used, i. e. the second Receive-PDO is configured via object 1401h, the third Receive-PDO via object 1402h, etc.. For the Transmit-PDOs this are objects 1801h, 1802h, etc..

Maximum 512 Receive PDOs and 512 Transmit-PDOs are possible according to communication profile DS 301.

The CANopen Slave option module supports a maximum of 4 Receive-PDOs and 4 Transmit-PDOs.

Consequently, the

- Receive-PDOs 1, 2, 3 and 4 must be configured with objects 1400h, 1401h, 1402h and 1403h

and the

- Transmit-PDOs 1, 2, 3 and 4 must be configured with objects 1800h, 1801h, 1802h and 1803h.

The values of the objects have the following meanings according to communication profile DS 301:

Receive-PDOs

Object	Meaning
1400h (- 15FF)	Communication parameter for the first Receive-PDO of a device.
Subindex 0	Number of the subindex, which contains the last valid entry
Subindex 1	COB ID of the PDO and validity
Subindex 2	Transfer type of the PDO: synchronized or asynchronous
Subindex 3	*Not used*
Subindex 4	Compatibility entry. Optional.
Subindex 5	*Not used.*

Transmit-PDOs

Object	Meaning
1800h (- 19FF)	Communication parameter for the first Transmit-PDO of a device.
Subindex 0	Number of the subindex, which contains the last valid entry
Subindex 1	COB ID of the PDO and validity
Subindex 2	Transfer type of the PDO: synchronized, asynchronous, event-driven.
Subindex 3	Inhibit time of the PDO. Optional
Subindex 4	Compatibility entry. Optional.
Subindex 5	Timer value, if the PDO transfer is time-controlled. Optional.

We will consider the meaning of the subindices in more detail.

Object 1400h / 1800h Subindex 0 - Number of valid subindices

Subindex 0 indicates the number of the last valid subindex. The value is at least 2 and increases according to the optional further entries supported (subindices). The value is generally "read only", so only of interest to you for information purposes.

Object 1400h / 1800h Subindex 1 - COB ID of the PDO

Subindex 1 contains various information. The value has data type Unsigned32 and is organized as follows:

Bit	Value / meaning
31 (MSB)	0: PDO exists and is valid 1: PDO does not exist or is not valid According to the predefined identifier assignment (see chapter ▷Data exchange and objects of the physical bus system ◁ from page 18 onward), only 4 PDOs are possible for writing and 4 PDOs for reading. If a device supports more than these four PDOs, you must assign the COB IDs for the additional PDOs yourself (bits 10 - 0 in this subindex). For this reason, bit 31 of the additional PDOs usually has a value of 1 and users do not set it to 0 until they have assigned a COB ID.
30	0: The PDO can be requested by means of a remote request 1: The PDO cannot be requested by means of a remote request
29	0: 11-bit CAN ID 1: 29-bit CAN ID (is not supported by the CANopen Slave option module)
28 - 11	(is not supported by the CANopen Slave option module)
10 - 0 (LSB)	COB ID of the PDO. In the default settings of a device, the first four PDOs for writing and reading have the COB IDs after the predefined identifier assignment. (see chapter ▷Data exchange and objects of the physical bus system ◁ from page 18 onward)

If you do not need more than 4 PDOs for writing and 4 PDOs for reading, this value will mainly only be of interest to you for information purposes.

Object 1400h / 1800h Subindex 2 - Transfer type of the PDO

This value is very important, as it indicates when a PDO is transmitted. The type of transfer is configured in Subindex 2. The following assignment applies for transmitting and receiving PDOs:

Subindex 2 / Type	**Effect** **Transmit-PDO 1800h**	**Receive-PDO 1400h**
0 Synchronized	The system sends after each received SYNC message frame but only if the transferred data are changed [a)]	The system applies the PDO with an appropriate COB ID that was received before the last SYNC message frame
1 - 240 Synchronized	The system sends after receiving the set number of SYNC message frames [a)]	The system applies the PDO with an appropriate COB ID that was received before the last SYNC message frame
252 Remote	On receiving a SYNC message frame, the system updates the PDO. The system does not send until it receives a remote request. [a)]	*Not possible*
253 Remote	The system updates and sends after receiving a remote request.	*Not possible*

Subindex 2 / Type	Effect Transmit-PDO 1800h	Receive-PDO 1400h
254 Asynchronous	The system sends on a manufacturer-specific basis. Note: With this type, most devices transfer the PDO on a time (when a timer runs down). The time is set under subindex 5. [b)]	The system applies each PDO with an appropriate COB ID when it is received
255 Asynchronous	The system sends in dependence on the supported device profile. Note: With this type, most devices transfer the PDO as soon as one of the mapped values has changed. [c)]	The system applies each PDO with an appropriate COB ID when it is received

a) See chapter ▹Synchronization of data exchange ◃ on page 67

b) Time-controlled transmission means that the transmit condition is linked to a timer. This timer is set for the first Transmit-PDO by means of subindex 5 in object 1800h (16 bit). The resolution is in milliseconds. The timer is started when the state changes to OPERATIONAL. The PDO is then sent cyclically at the times set in the timer. The timer is deleted by writing the value "0" in Subindex 5. Time-controlled reception does not exist, but all PDOs with appropriate COB ID are applied.

c) Event-driven transmission means that the transmission requirement is linked to the change in value of the mapped objects. If for example 3 objects are mapped, the PDO is transmitted as soon as at least one of the three values changes. If the values remain constant, no PDO will be transmitted. Because of this, bus loading can be reduced (message frames are only transmitted when they contain new information). Event-driven reception means that all PDOs with appropriate COB ID are applied.

Object 1400h / 1800h Subindex 3 - Inhibit time of the PDO

Using subindex 3, you set a PDO inhibit time. This does not extend the response time at the relative first value change; rather, it is active on the changes that directly follow this one. The inhibit time describes the minimum period of time that the system must wait between sending two of the same message frames. The value has no meaning for Receive-PDOs.

Object 1400h / 1800h Subindex 4

The value of subindex 4 has no relevant meaning.

Object 1400h / 1800h Subindex 5 - Time for event-driven PDO transfer

The value of subindex 5 indicates the time in ms for sending the PDO (see Subindex 2). The value has no meaning for Receive-PDOs.

Transfer properties of the PDOs - default setting

The CANopen Slave option module is configured as follows, by default:

Object 1400	Value
Subindex 0	2
Subindex 1	200 + Node ID
Subindex 2	255

I. e. RxPDO 1 must have the identifier "200 + Node ID of the CANopen Slave option module" and this PDO is evaluated asynchronously (the SYNC message frame has no effect).

Object 1401	**Value**
Subindex 0	2
Subindex 1	300 + Node ID
Subindex 2	255

I. e. RxPDO 1 must have the identifier "300 + Node ID of the CANopen Slave option module" and this PDO is evaluated asynchronously (the SYNC message frame has no effect).

Object 1402	**Value**
Subindex 0	2
Subindex 1	400 + Node ID
Subindex 2	255

I. e. RxPDO 1 must have the identifier "400 + Node ID of the CANopen Slave option module" and this PDO is evaluated asynchronously (the SYNC message frame has no effect).

Object 1403	**Value**
Subindex 0	2
Subindex 1	500 + Node ID
Subindex 2	255

I. e. RxPDO 1 must have the identifier "500 + Node ID of the CANopen Slave option module" and this PDO is evaluated asynchronously (the SYNC message frame has no effect).

Object 1800	**Value**
Subindex 0	5
Subindex 1	180 + Node ID
Subindex 2	255
Subindex 3	0
Subindex 4	0
Subindex 5	0

I.e. TxPDO 1 has the identifier "180 + Node ID of the CANopen Slave option module" and this PDO is transmitted asynchronously (the SYNC message frame has no effect). The system sends whenever an object mapped in this TxPDO is changed.

Object 1801	**Value**
Subindex 0	5
Subindex 1	280 + Node ID
Subindex 2	255

Object 1801	Value
Subindex 3	0
Subindex 4	0
Subindex 5	0

I.e. TxPDO 1 has the identifier "280 + Node ID of the CANopen Slave option module" and this PDO is transmitted asynchronously (the SYNC message frame has no effect). The system sends whenever an object mapped in this TxPDO is changed.

Object 1802	Value
Subindex 0	5
Subindex 1	380 + Node ID
Subindex 2	255
Subindex 3	0
Subindex 4	0
Subindex 5	0

I.e. TxPDO 1 has the identifier "380 + Node ID of the CANopen Slave option module" and this PDO is transmitted asynchronously (the SYNC message frame has no effect). The system sends whenever an object mapped in this TxPDO is changed.

Object 1804	Value
Subindex 0	5
Subindex 1	480 + Node ID
Subindex 2	255
Subindex 3	0
Subindex 4	0
Subindex 5	0

I.e. TxPDO 1 has the identifier "480 + Node ID of the CANopen Slave option module" and this PDO is transmitted asynchronously (the SYNC message frame has no effect). The system sends whenever an object mapped in this TxPDO is changed.

Transfer properties of the PDOs - example setting

In our example, the following setup was configured:

- that objects A200Sub1 and A200Sub2 are transferred from the b maXX PLC to the CANopen Slave option module and must be sent to the CANopen Master (→ ro (read only); see "Creating variables", variable "ud_MyReadVar1" and "ud_MyReadVar2")
- that objects A680Sub1 and A580Sub1 must be received from the CANopen Master and are transferred from the CANopen Slave option module to the b maXX PLC (→ rw (read and write); see "Creating variables", variable "ud_MyWriteVar1" and "u_MyWriteVar2")

We will transmit the objects A200Sub1 and A200Sub2 in TxPDO1 to the CANopen Master and receive the objects A680Sub1 and A580Sub1 in RxPDO1 from the CANopen Master.

For our example, the default settings for the transfer properties of TxPDO1 (in object 1400) and RxPDO1 (in object 1800) are not changed, i. e. you do not need to change these objects from the CANopen Master via "Write SDO".

The transfer properties of TxPDO1, TxPDO2 and TxPDO3 (Objects 1401, 1402 and 1403) as well as RxPDO1, RxPDO2 and RxPDO3 (Objects 1801, 1802 and 1803) are not relevant for our example, as these PDOs are not used.

The following setting thus results with Node ID 5:

Object 1400	**Value**
Subindex 0	2
Subindex 1	205
Subindex 2	255

I.e. RxPDO 1 has the identifier "205" and this PDO is evaluated asynchronously (the SYNC message frame has no effect).

Object 1800	**Value**
Subindex 0	5
Subindex 1	185
Subindex 2	255
Subindex 3	0
Subindex 4	0
Subindex 5	0

I.e. TxPDO 1 has the identifier "185" and this PDO is transmitted asynchronously (the SYNC message frame has no effect). The system sends whenever an object mapped in this TxPDO is changed.

Mapping of objects in a PDO

Mapping of objects in a PDO must be defined as the second configuration step for data exchange. There are also special, communication profile specific objects for the mapping, in which this is written for a network node. In the following, the configuration object is written for the first Receive-PDO (1600h) and the first Transmit-PDO (1A00h). To configure all further PDOs, the next higher object index must be used, i. e. the mapping of the second Receive-PDO is configured via object 1601h, the third Receive-PDO via object 1602h etc..

The CANopen Slave option module supports a maximum of 4 Receive-PDOs and 4 Transmit-PDOs.

Consequently, the mapping must be defined for the

- Receive-PDOs 1, 2, 3 and 4 must be configured with objects 1600h, 1601h, 1602h and 1603h

and for the

- Transmit-PDOs 1, 2, 3 and 4 with objects 1A00h, 1A01h, 1A02h and 1A03h.

In detail, the objects according to DS 301 have the following meaning:

Receive-PDOs

Object	Meaning
1600h (- 17FF)	Mapping parameter for the first Receive-PDO of a device.
Subindex 0	Number of mapped objects
Subindex 1 - Subindex 64	Information on the mapped object

Transmit-PDOs

Object	Meaning
1A00h (- 1BFF)	Mapping parameter for the first Transmit-PDO of a device.
Subindex 0	Number of mapped objects
Subindex 1 - Subindex 64	Information on the mapped object

We will consider the meaning of the subindices in more detail.

Object 1600h / 1A00h Subindex 0 - Number of mapped objects

Subindex 0 indicates the number of objects mapped in this PDO. If you wish to change the mapping of a PDO, you must first of all set the number of mapped objects to zero. You can then change the mapping, and then set the number of mapped objects in subindex 0. Please note that device-dependent limitations can exist here with regard to the number of objects to be mapped.

The CANopen Slave option module supports a maximum of 8 mapped objects per PDO.

Object 1600h / 1A00h Subindex 1 to 64 - Mapping information

Subindices 1 to 64 contain the mapping information of the objects in a PDO. Subindex 1 for the first mapped object, subindex 2 for the second mapped object etc.. The data type of these subindices is Unsigned32. The contents have the following meaning:

Bit 31 - 16	Bit 15 - 8	Bit 7 - 0
Index of the mapped object	Subindex of the mapped object	Number of bits of the mapped object

The system maps the objects successively in one PDO.

NOTE

Please note that you can define which objects are mappable in PDOs under ▷Creating variables ◁ on page 34.

A default mapping is defined for the CANopen Slave option module. If these default mapping values suffice, you do not need to make any further settings here. I. e. you do not need to change the default mapping (from the CANopen Master with "Write SDO").

Mapping properties of the PDOs - default setting

The following mapping is set as default on the CANopen Slave option module:

RxPDOs:

Object 1600	Value
Subindex 0	5
Subindex 1	A4800108h
Subindex 2	A4C00108h
Subindex 3	A5400110h
Subindex 4	A5800110h
Subindex 5	A5800210h

I. e. RxPDO 1 contains five objects, which are entered in the object entries:
Object index A480, Object subindex 01, Number of bits: 8 (Data type: INTEGER8);
Object index A4C0, Object subindex 01, Number of bits: 8 (Data type: UNSIGNED8);
Object index A540, Object subindex 01, Number of bits: 16 (Data type: INTEGER16);
Object index A580, Object subindex 01, Number of bits: 16 (Data type: UNSIGNED16);
Object index A580, Object subindex 02, Number of bits: 16 (Data type: UNSIGNED16);

Object 1601	Value
Subindex 0	2
Subindex 1	A6400120h
Subindex 2	A6800120h

I.e. RxPDO 2 contains two objects, which are entered in the object entries:
Object index A640, Object subindex 01, Number of bits: 32 (Data type: INTEGER32);
Object index A680, Object subindex 01, Number of bits: 32 (Data type: UNSIGNED32);

Object 1602	Value
Subindex 0	4
Subindex 1	A4800108h
Subindex 2	A4C00108h
Subindex 3	A5400110h
Subindex 4	A6800120h

I. e. RxPDO 3 contains four objects, which are entered in the object entries:
Object index A480, Object subindex 01, Number of bits: 8 (Data type: INTEGER8);
Object index A4C0, Object subindex 01, Number of bits: 8 (Data type: UNSIGNED8);
Object index A540, Object subindex 01, Number of bits: 16 (Data type: INTEGER16);
Object index A680, Object subindex 01, Number of bits: 32 (Data type: UNSIGNED32);

Object 1603	Value
Subindex 0	3
Subindex 1	A5400110h
Subindex 3	A5800110h
Subindex 4	A6800120h

I. e. RxPDO 4 contains three objects, which are entered in the object entries:
Object index A540, Object subindex 01, Number of bits: 16 (Data type: INTEGER16);
Object index A580, Object subindex 01, Number of bits: 16 (Data type: UNSIGNED16);
Object index A680, Object subindex 01, Number of bits: 32 (Data type: UNSIGNED32);

TxPDOs:

Object 1A00	Value
Subindex 0	5
Subindex 1	A0000108h
Subindex 2	A0400108h
Subindex 3	A0C00110h
Subindex 4	A1000110h
Subindex 5	A1000210h

I.e. TxPDO 1 contains five objects, which are removed from the object entries:
Object index A000, Object subindex 01, Number of bits: 8 (Data type: INTEGER8);
Object index A040, Object subindex 01, Number of bits: 8 (Data type: UNSIGNED8);
Object index A0C0, Object subindex 01, Number of bits: 16 (Data type: INTEGER16);
Object index A100, Object subindex 01, Number of bits: 16 (Data type: UNSIGNED16);
Object index A100, Object subindex 02, Number of bits: 16 (Data type: UNSIGNED16);

Object 1A01	Value
Subindex 0	2
Subindex 1	A1C00120h
Subindex 2	A2000120h

I.e. TxPDO 2 contains two objects, which are removed from the object entries:
Object index A1C0, Object subindex 01, Number of bits: 32 (Data type: INTEGER32);
Object index A200, Object subindex 01, Number of bits: 32 (Data type: UNSIGNED32);

Object 1A02	Value
Subindex 0	4
Subindex 1	A0000108h
Subindex 2	A0400108h
Subindex 3	A0C00110h
Subindex 4	A2000120h

I.e. TxPDO 3 contains four objects, which are removed from the object entries:
Object index A000, Object subindex 01, Number of bits: 8 (Data type: INTEGER8);
Object index A040, Object subindex 01, Number of bits: 8 (Data type: UNSIGNED8);
Object index A0C0, Object subindex 01, Number of bits: 16 (Data type: INTEGER16);
Object index A200, Object subindex 01, Number of bits: 32 (Data type: UNSIGNED32);

Object 1A03	Value
Subindex 0	3
Subindex 1	A0C00110h
Subindex 3	A1000110h
Subindex 4	A2000120h

I.e. TxPDO 4 contains three objects, which are removed from the object entries:
Object index A0C0, Object subindex 01, Number of bits: 16 (Data type: INTEGER16);
Object index A100, Object subindex 01, Number of bits: 16 (Data type: UNSIGNED16);
Object index A200, Object subindex 01, Number of bits: 32 (Data type: UNSIGNED32);

Mapping properties of the PDOs - example setting

In our example, the following setup was configured:

- that objects A200Sub1 and A200Sub2 are transferred from the b maXX PLC to the CANopen Slave option module and must be sent to the CANopen Master (→ ro (read only); see "Creating variables", variable "ud_MyReadVar1" and "ud_MyReadVar2")
- that objects A680Sub1 and A580Sub1 must be received from the CANopen Master and are transferred from the CANopen Slave option module to the b maXX PLC (→ rw (read and write); see "Creating variables", variable "ud_MyWriteVar1" and "u_MyWriteVar2")

We will transmit the objects A200Sub1 and A200Sub2 to the CANopen Master and receive the objects A680Sub1 and A580Sub1 from the CANopen Master.

The default mapping for the TxPDOs does not contain any TxPDO which includes the objects A200Sub1 and A200Sub2. Therefore, the mapping characteristics of a TxPDO (taking TxPDO1) must be changed.

The default mapping for the RxPDOs contains RxPDO4, which includes the objects A580Sub1 and A680Sub1. We use RxPDO4 and do not need to change the mapping properties.

We only need to note that the value for object A540Sub1 is also transmitted in RxPDO4, and object A580Sub1 must not be entered in data byte 0 and 1 but in data byte 3 and 4 on the CANopen Master.

We have not assigned any variable to object A540Sub1 in the CANopen Slave Configurator, therefore do not use the content of the object in the PROPROG wt II project.

The mapping properties of TxPDO1 (object 1A00) are changed by changing this object from the CANopen Master with "Write SDO" (after each activation of the CANopen Slave). Please take the values (note sequence!) from the table.

SDO write			
Object	**Subindex**	**Value**	**Note**
1A00	0	0	Delete entries
1A00	1	16#A2000120	1st object: Object 16#A200, Subindex 16#01, Length 16#20 (32 bit)
1A00	2	16#A2000220	2nd object: Object 16#A200, Subindex 16#02, Length 16#20 (32 bit)
1A00	0	2	2 entries for TxPDO1

The following setting thus results:

Object 1603	**Value**
Subindex 0	3
Subindex 1	A5400110h
Subindex 3	A5800110h
Subindex 4	A6800120h

I. e. RxPDO 4 contains three objects, which are entered in the object entries:
Object index A540, Object subindex 01, Number of bits: 16 (Data type: INTEGER16);
Object index A580, Object subindex 01, Number of bits: 16 (Data type: UNSIGNED16);
Object index A680, Object subindex 01, Number of bits: 32 (Data type: UNSIGNED32);

TxPDOs:

Object 1A00	Value
Subindex 0	2
Subindex 1	A2000120h
Subindex 2	A2000220h

I.e. TxPDO 1 contains two objects, which are removed from the object entries:
Object index A200, Object subindex 01, Number of bits: 32 (Data type: UNSIGNED32);
Object index A200, Object subindex 02, Number of bits: 32 (Data type: UNSIGNED32);

Reading objects via PDO

Process data exchange is limited on the PROPROG wt II side to reading or writing global variables, which you have created with the CANopen Slave Configurator under ▹Creating variables ◃ on page 34, and to which you have assigned CANopen objects.

If a CANopen Master is to read objects via PDO, then the CANopen Slave must transmit this PDO as a TxPDO. This TxPDO contains the objects.

The variables that are assigned to these objects must be provided with a value in the application program on the b maXX PLC.

I. e. the variables are **written** in the application program on the b maXX PLC.

NOTE

Process data exchange via PDO is only possible if the CANopen Slave is in OPERATIONAL NMT state. In this connection, also see ▹Network commands - NMT ◃ on page 46.

In the example, the variable "ud_MyReadVar1" was created in the CANopen Slave Configurator. The variable was also assigned to object index A200h and object subindex 1. In addition, the variable was given the initial value 1000. "ud_MyReadVar2" (with object index A200h and object subindex 2 as well as initial value 2000) was also created.

A CANopen Master can now request the TxPDO1 from the CANopen Slave.

The value 16#0000_03E8_0000_07D0 must be displayed in the CANopen Master
(16#0000_03E8 = 1000 = value of "ud_MyReadVar1;
16#0000_07D0 = 2000 = value of "ud_MyReadVar2).

Writing objects via PDO

Process data exchange is limited on the PROPROG wt II side to reading or writing global variables, which you have created with the CANopen Slave Configurator under ▹Creating variables ◃ on page 34, and to which you have assigned CANopen objects.

If a CANopen Master is to write to objects via PDO, the CANopen Slave must receive this PDO as a RxPDO. This RxPDO contains the objects.

The variables that are assigned to these objects must be read in the application program on the b maXX PLC.

I. e. the variables are **read** in the application program on the b maXX PLC.

NOTE

Process data exchange via PDO is only possible if the CANopen Slave is in OPERATIONAL NMT state. In this connection, also see ▹Network commands - NMT ◃ on page 46.

In the example, the variable "ud_MyWriteVar1" was created in the CANopen Slave Configurator. The variable was also assigned to object index A680h and object subindex 1. In addition, the variable was given the initial value 1500 and the CANopen object limited to the value range 1000 <= x <= 2000. In addition, the variable "u_MyWriteVar2" was created (with object index A580 and object subindex 1, without initial value or limitation).

A CANopen Master can now write to the RxPDO4 of the CANopen Slave.

A value between 1000 and 2000 must be written in the CANopen Master for "ud_MyWriteVar1", and a value (according to the data type) between 0 and 65535 for "u_MyWriteVar2".

For example:

16#223D_xxxx = 8765 = value for "u_MyWriteVar2" (in byte 3 and 4); 0 <= 8765 <= 65535 (xxxx is the value which is written to object A540Sub1)

16#0000_04D2 = 1234 = value for "ud_MyWriteVar1"; 1000 <= 1234 <= 2000

The CANopen Master can (but does not have to) take the limitation of the objects from the EDS file (see ▹Creating variables ◃ on page 34 and ▹Generating an EDS file ◃ on page 42). The user may need to implement monitoring of such limitations in the application program on the b maXX PLC.

In the CANopen Slave option module, the previous value "1500" of object A680Sub1 is then overwritten with "1234" and the previous value "0" of object A580Sub1 is overwritten with "8765".

In the application program on the b maXX PLC, the previous value 1500 of "ud_MyWriteVar1" is therefore overwritten with 1234 and the previous value 0 of "u_MyWriteVar2" is overwritten with 8765.

3.3.7.6 Synchronization of data exchange

Definition according to CANopen specification

With many applications, it is useful to synchronize the acceptance of input data and the transmission of output data. CANopen makes the SYNC message frame available for this purpose. The SYNC message frame has the highest priority after the NMT message frame, and no user data. The transfer of the SYNC message frame occurs according to the consumer/producer communication model, i. e. a network node generates a SYNC message frame, which is processed by one or several other network nodes. There is no confirmation of the receipt of the message frame. Receipt acts as a trigger for the network nodes to read received PDOs and to send PDOs, provided that a PDO is configured for this transfer type (see Chapter ▹Configuration of the transfer properties of a PDO ◃ on page 54). The setting for whether a network node is a consumer or producer occurs via object 1005h:

SYNC message frame:

Object	Meaning
1005h	Configuration of the SYNC message frame.
Subindex 0	COB ID and validity

The value of subindex 0 has data type Unsigned32 and is organized as follows:

Bit	Value / meaning
31 (MSB)	*Not relevant*
30	0: Network node does not generate SYNC message frames 1: Network node generates SYNC message frames (not supported by the CANopen Slave option module)
29	0: 11-bit CAN ID 1: 29-bit CAN ID (is not supported by the CANopen Slave option module)
28 - 11	Is not supported by the CANopen Slave option module
10 - 0 (LSB)	COB ID of the SYNC message frame. As the CANopen Slave option module uses the COB ID according to the predefined identifier assignment (see Chapter ▷Data exchange and objects of the physical bus system ◁ from page 18 onward), this must be 80h.

NOTE

The CANopen Slave option module can only be operated as SYNC consumer. I. e. another network node must be SYNC producer.

The CANopen Slave option module for b maXX PLC is set as SYNC consumer with COB ID 80h (according to the predefined identifier assignment) as default setting. I. e. the object 1005h has the content ’00000080h’.

Receipt of an SYNC message frame generates an interrupt on the CANopen Slave option module. This is forwarded to the b maXX PLC and the b maXX controller. Because of this, this signal can be used to synchronize the b maXX 4400. All relevant message frames (synchronized PDOs) must be transmitted to all configured slaves within one SYNC interval (communication cycle). Transfer rate, cable length, number of nodes, size of message frames and processing times on the CANopen Slave module are to be taken into consideration during this.

The acceptance and transfer of PDO contents occurs according to the following scheme in a SYNC message frame:

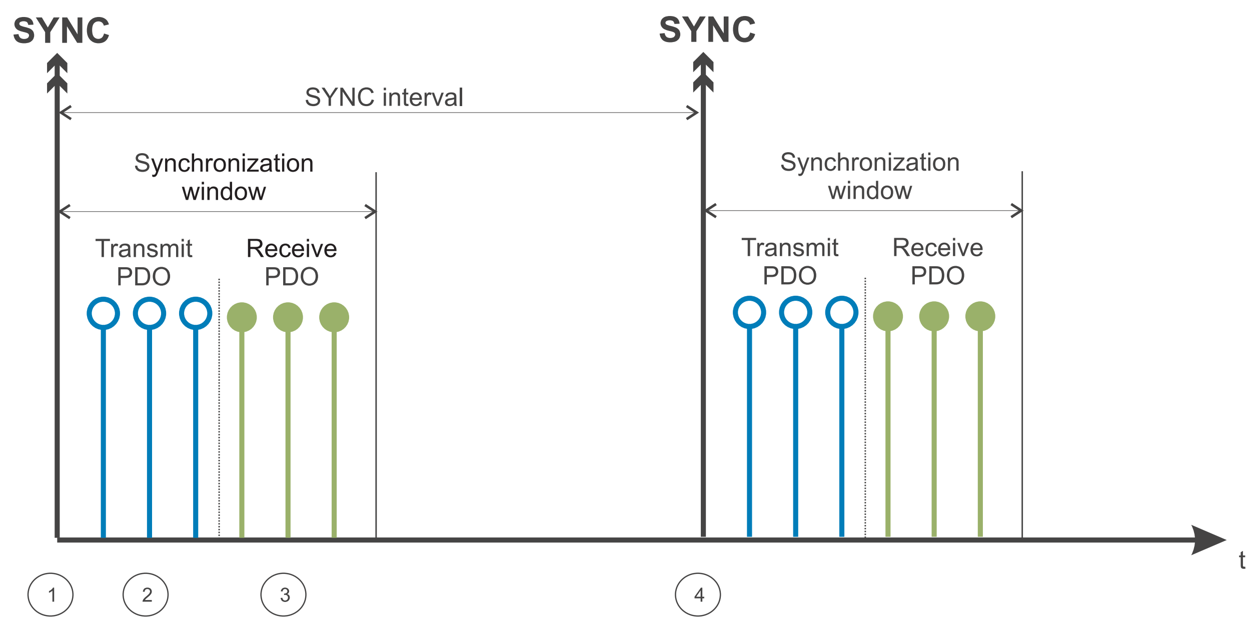


Figure 29: Synchronization mechanism

After receipt of the SYNC message frame ①, a network node first of all transmits the Transmit-PDOs (TxPDOs) ②. The PDOs received up to that point (RxPDOs) ③ are then transferred to a buffer memory. Processing of the RxPDOs only occurs after receipt of the next SYNC message frame ④. Synchronized PDOs may only occur within the synchronization window.

For the CANopen Slave option module, this means:
After receipt of the SYNC message frame, ① the CANopen Slave option module transmits the Transmit-PDOs (TxPDOs) ②. The PDOs received up to that point (RxPDOs) ③ are then transferred to a buffer memory. Copying of the data of these RxPDOs to the DPRAM of the b maXX PLC only occurs after receipt of the next SYNC message frame ④.

The CANopen Slave option module offers the option of monitoring the SYNC interval. The SYNC interval (as a multiple of 1 µs) must be set in the object 1006h on the CANopen Slave option module. The CANopen Slave option module reports the failure of a SYNC message frame to the application program on the b maXX PLC.

SYNC interval:

Object	**Meaning**
1006h	SYNC interval / length of communication cycle.
Subindex 0	0: not used n: Length of the SYNC interval in µs.

The values 1000, 2000, 4000 and 8000 can be set on the CANopen Slave option module. With the multiplier of 1 µs, the SYNC intervals 1 ms, 2 ms, 4 ms or 8 ms thus result. Default setting is 0, i. e. no SYNC interval is set and therefore the receipt of the SYNC message frame is not monitored.

With the transfer type of the PDO (see Chapter ▷Configuration of the transfer properties of a PDO ◁ on page 54), you can also configure the setup so that a PDO is only transferred after the nth SYNC message frame. Short cycles can thus be set for important data and long cycles for less important data.

Detecting a SYNC message frame failure

The failure of a SYNC message frame is reported to the application program on the b maXX PLC via the FB CANop405_KERNEL_STATE_SL. In this connection, please see Chapter ▷CANopen-Kernel-State ◁ on page 73 and the online help for the FB CANop405_KERNEL_STATE_SL.

Monitoring is activated with the receipt of the first SYNC message frame. If no SYNC message frame is received after expiry of the SYNC interval (object 1006h), the failure of the SYNC message frame is indicated by the FB CANop405_KERNEL_STATE_SL and the error message "SYNC frame failure". As soon as SYNC message frames are received again within the SYNC interval, the error message disappears.

NOTE

The SYNC monitoring is switched off as default setting of object 1006h (SYNC interval), i. e. with default setting of object 1006h the FB CANop405_KERNEL_STATE_SL will not report a SYNC failure.

3.3.7.7 Node Guarding

Definition according to CANopen specification

The mechanisms Node Guarding and Life Guarding are available for failure monitoring of network nodes. The network nodes are monitored by the CANopen Master by means of Node Guarding, and can themselves detect the failure of the CANopen Master via Life Guarding. Both mechanisms are based on the same functionality:

During Node Guarding, the CANopen Master requests a response from the CANopen Slave at defined ’Guard Time’ intervals via a remote message frame. The Slave responds with the Guarding message. This contains the NMT state (see Chapter ▷Network commands - NMT ◁ on page 46) of the CANopen Slave and a toggle bit, which must change after each message. If state or toggle bit do not agree with that expected by the CANopen Master, or if no response occurs within a certain period of time, the ’Lifetime’, the Master assumes that there is a Slave error. Vice-versa, the Slave can detect failure of the Master. If the Slave does not receive a message request from the Master within the set ’Lifetime’, it assumes a Master failure, reports the failure to the application program on the b maXX PLC and transmits an emergency message frame.

Node Guarding is switched off as default setting on the CANopen Slave option module.

Two objects are important on the CANopen Slave option module for setting the Guarding functionality; these must be set by the CANopen Master via SDO:

Guarding Time:

Object	Meaning
100Ch	Guarding Time configuration.
Subindex 0	0: Guarding is not active n: The distance between two guard message frames in ms

Lifetime Factor:

Object	Meaning
100Dh	Lifetime Factor configuration.
Subindex 0	0: Life Guarding is not active n: The Lifetime results from n x Guard Time

Default setting for Guarding Time is 0, i. e. no Guarding Time is set.

Default setting for Lifetime Factor is 0, i. e. no Lifetime Factor is set.

Node Guarding is available in all communication phases (STOPPED, PRE-OPERATIONAL, OPERATIONAL). The toggle bit is only reset to its default value in the INITIALIZATION phase. This means that the toggle mechanism is continued also on changing state. Node Guarding is started in the CANopen Slave option module after receipt of the first Guarding request message frame (which the CANopen Master transmits). From this moment, the monitoring time parameterized in objects 100Ch and 100Dh runs in the option module CANopen slave.

Detecting a Node Guarding failure

"Failure of a Node Guarding message frame" is reported to the application program on the b maXX PLC via the FB CANop405_LIFE_GUARD_SL. In this connection please see the online help for the FB CANop405_LIFE_GUARD_SL.

3.3.7.8 Heartbeat

In addition to the Node Guarding and Life Guarding mechanisms, the heartbeat mechanism is also available for failure monitoring of network nodes.

With the heartbeat, one or several heartbeat producers sends heartbeat message frames at defined "Heartbeat Producer Time" intervals. A heartbeat consumer monitors whether the heartbeat message frames were sent within the "Heartbeat Consumer Time". A heartbeat consumer can thus monitor up to 127 heartbeat producers. Up to 127 different "Heartbeat Consumer Times" can be set on the heartbeat consumer.

The CANopen Slave option module for b maXX PLC can be set as heartbeat producer.

NOTE

The CANopen Slave option module can only be set as heartbeat producer.

It can NOT be set as heartbeat consumer.

The setting of the heartbeat producer functionality on the CANopen Slave option module occurs via the object Producer Heartbeat Time (object 1017h), which must be set via the CANopen Master.

Producer Heartbeat Time

Object	Meaning
1017	Producer Heartbeat Time for the CANopen Slave option module
Subindex 0	Heartbeat Time

The Producer Heartbeat Time is specified as a multiple of 1 ms. The entry is writeable and readable and of data type UNSIGNED16.

Object 1017h (subindex 0) has default value "0" in the CANopen Slave option module, i. e. the CANopen Slave option module does not transmit any heartbeat message frames. The value for object 1017h (subindex 0) can be changed by a CANopen Master via write SDO.

3.3.7.9 Own network node address (Local Node ID)

With the CANopen Slave option module, you can set the Node ID either via the DIP switches or by means of software during initialization. It can be necessary for an application program to read out the CANopen Slave's own Node ID.

The CANopen Slave option module's own Node ID is read out with FB CANop405_LOC_NODE_ID_SL.

In this connection, please also see the online help for the FB CANop405_LOC_NODE_ID_SL.

Determining the Local Node ID

You can read the CANopen Slave option module's own network node address (Local Node ID) in the application program on the b maXX PLC with the FBs CANop405_LOC_NODE_ID_SL.

To do this, please follow the steps below

- Create a POU with PLC type SH03_30 and processor type BM4_O_PLC01. This POU is to be subsequently called up in a cyclical task.
  Alternatively, you can also use the POU in which the NMT state is read.
- Place the FB CANop405_LOC_NODE_ID_SL in this POU.
- Interconnect the function block with variables of the correct data type.
- Create a cyclical task with average to low priority, if this is not already present in your project. Link the created POU in this task with FB CANop405_LOC_NODE_ID_SL.

If you use the POU in which the NMT state is read, you have already assigned this POU to a task.

- Compile the project and send it to the PLC as (boot) project.
- Start the project.

The CANopen Slave's own network node address (Local Node ID) is read with x_ENABLE = 1. The FB CANop405_LOC_NODE_ID_SL reports successful execution with x_CONFIRM = 1.

Example: Determine Local Node ID

In order to be able to use the FB CANop405_LOC_NODE_ID_SL, we create a POU and place the function block in it. We assign the generated POU a cyclical task with 200 ms call interval. After interconnecting the function block, compile the project and send it as project to the b maXX PLC. Start the project. The CANopen Slave's own network node address (Local Node ID) is read with x_ENABLE = 1 ("5" in our case):

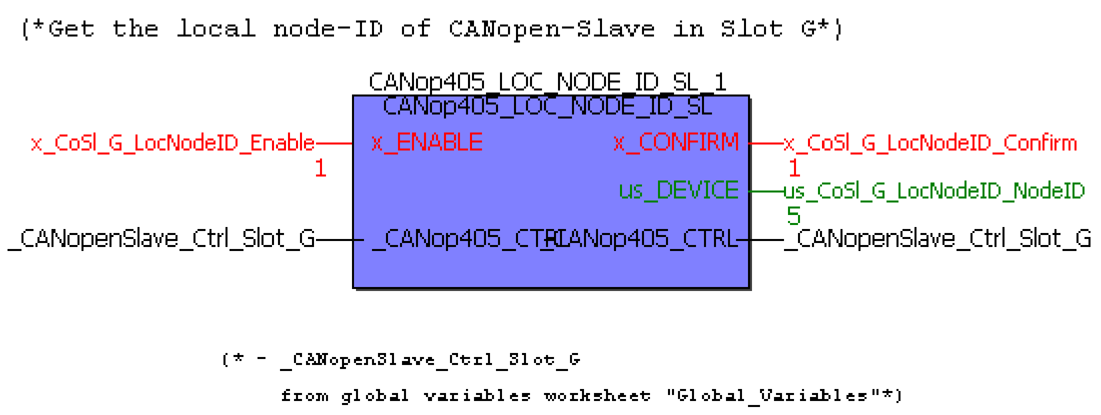

(*Get the local node-ID of CANopen-Slave in Slot G*)
CANop405_LOC_NODE_ID_SL_1
CANop405_LOC_NODE_ID_SL
x_CoSl_G_LocNodeID_Enable
1
x_ENABLE
x_CONFIRM
x_CoSl_G_LocNodeID_Confirm
1
us_DEVICE
us_CoSl_G_LocNodeID_NodeID
5
_CANopenSlave_Ctrl_Slot_G
_CANop405_CTRL
_CANop405_CTRL
_CANopenSlave_Ctrl_Slot_G
(* - _CANopenSlave_Ctrl_Slot_G
from global variables worksheet "Global_Variables"*)

Figure 30: Determining the CANopen Slave's own network node address (Local Node ID)

3.3.7.10 CANopen-Kernel-State

The CANopen-Kernel-State of the CANopen Slave option module is read out with FB CANop405_KERNEL_STATE_SL.

In this connection, please also see the online help for the FB CANop405_KERNEL_STATE_SL.

Determine CANopen-Kernel-State

You can read the CANopen-Kernel-State of the CANopen Slave option module in the application program on the b maXX PLC with the FBs CANop405_KERNEL_STATE_SL.

To do this, please follow the steps below:

- Create a POU with PLC type SH03_30 and processor type BM4_O_PLC01. This POU is to be subsequently called up in a cyclical task.

Alternatively, you can also use the POU in which the NMT state is read.

- Place the FB CANop405_KERNEL_STATE_SL in this POU.
- Interconnect the function block with variables of the correct data type.
- Create a cyclical task with average to low priority, if this is not already present in your project. Link the created POU in this task with FB CANop405_KERNEL_STATE_SL.

If you use the POU in which the NMT state is read, you have already assigned this POU to a task.

- Compile the project and send it to the PLC as (boot) project.
- Start the project.

The CANopen-Kernel-State of the CANopen Slave is read with x_ENABLE = 1. The FB CANop405_KERNEL_STATE_SL reports successful execution with x_CONFIRM = 1.

Example: Determine CANopen-Kernel-State

In order to be able to use the FB CANop405_KERNEL_STATE_SL, we create a POU and place the function block in it. We assign the generated POU a cyclical task with 200 ms call interval. After interconnecting the function block, compile the project and send it as project to the b maXX PLC. Start the project. The CANopen-Kernel-State of the CANopen Slave (in our case at output w_STATE "16#0000" or at output s_STATE 'No Error') is read with x_ENABLE = 1:

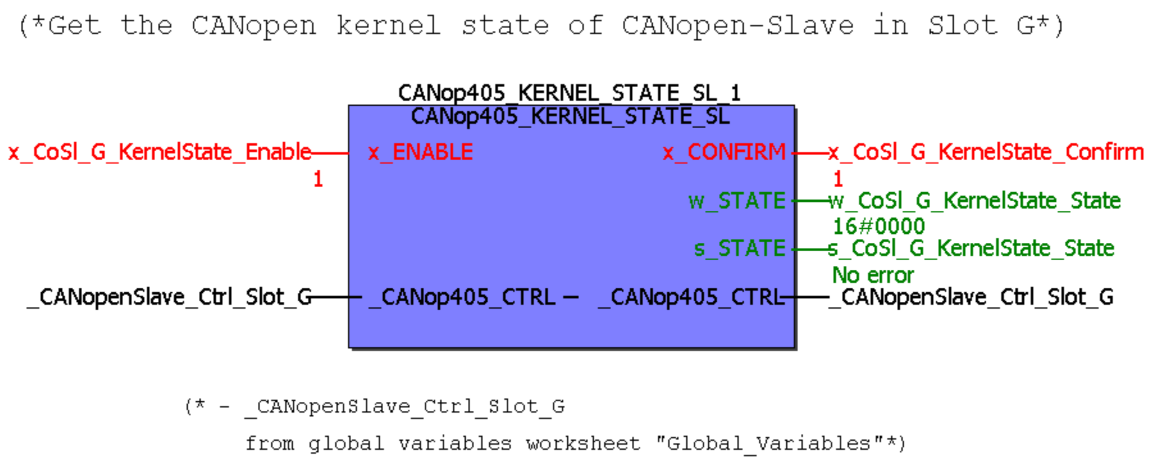

(*Get the CANopen kernel state of CANopen-Slave in Slot G*)
CANop405_KERNEL_STATE_SL_1
CANop405_KERNEL_STATE_SL
x_CoSl_G_KernelState_Enable
1
x_ENABLE
x_CONFIRM
x_CoSl_G_KernelState_Confirm
1
w_STATE
w_CoSl_G_KernelState_State
16#0000
s_STATE
s_CoSl_G_KernelState_State
No error
_CANopenSlave_Ctrl_Slot_G
_CANop405_CTRL
_CANop405_CTRL
_CANopenSlave_Ctrl_Slot_G
(* - _CANopenSlave_Ctrl_Slot_G
from global variables worksheet "Global_Variables"*)

Figure 31: Determining the CANopen-Kernel-State of the CANopen Slave

3.3.7.11 Data consistency

For data consistency, a distinction must be made between

- Consistency of data between CANopen Master and CANopen Slave option module and
- Consistency of data between CANopen Slave option module and b maXX PLC.

CANopen Master ↔CANopen Slave option module

The CANopen Slave option module supports consistency over all created variables (8-bit, 16-bit and 32-bit), i.e. the data in the CANopen message frame are consistently transferred.

CANopen Slave option module ↔ b maXX PLC

The b maXX PLC supports consistency for 8-bit values and for 16-bit values.

If consistency is desired over a greater data range (e. g. 32-bit values), a handshake mechanism is required between CANopen Slave option module and b maXX PLC.

For this you need the following code in a POU (in Structured Text (ST)):

Variable worksheet:

```
VAR
      u_tempAccessVar    :      UINT;  (* temporary use *)
      u_tempMerker       :      UINT;
END_VAR
VAR_EXTERNAL
      _CANopenSlave_Ctrl_Slot_G  :  CANopen_PLC_SL_BMSTRUCT; (*CANopen-Slave in
                                                                slot G*)
END_VAR
```

You can copy this code to the variables worksheet of your POU.

Code worksheet:

```
(* Diese Sequenz nur in einer Task einsetzen ! *)
(* Run this code sequence in only one task ! *)

_CANopenSlave_Ctrl_Slot_G.u_PLC_ACCESS  := UINT#1;  (* sperren / lock *)
u_tempAccessVar      :=  _CANopenSlave_Ctrl_Slot_G.u_OMC_ACCESS;

IF ((UINT_TO_WORD(u_tempAccessVar) AND WORD#16#0001) = WORD#16#0000) THEN
      (* wir dürfen zugreifen / we are allowed to access DPRAM *)
      IF (u_tempAccessVar <> u_tempMerker) THEN
      (* das Optionsmodul hat Daten ausgetauscht / data exchange done *)
            (* -- Fügen Sie hier Ihre Datenzugriffe ein -- *)
            (* ----- add your data access code here ----- *)
            ;
      END_IF;
END_IF;

u_tempMerker     :=  u_tempAccessVar;
_CANopenSlave_Ctrl_Slot_G.u_PLC_ACCESS  := UINT#0;  (* freigeben / unlock *)
```

You can copy this code to the code worksheet of your POU.

Data exchange (reading and writing variables) between b maXX PLC and CANopen Slave option module then takes place as follows, for example:

```
      (* --- Fügen Sie hier Ihre Datenzugriffe ein --- *)
      (* ----- add your data access code here ----- *)
      ud_MyWriteVar1 := ud_MyReadVar1  + UDINT#1;
```

NOTE

The code for consistent data exchange is only inserted in a POU and this POU may not be instantiated!

Otherwise the consistency of the data is not guaranteed.

3.4 The structure of CANopen message frames

3.4.1 General

Through the CANopen Slave option module and the function blocks from the library CANop405_PLC01_20bd03 (and higher), the message frame traffic is completely isolated and you need minimal knowledge of the structure of the individual CANopen message frames. However, if you wish to check the data traffic with a CAN analysis tool, you can take the structure of the individual message frames from the following chapters. The message frames are explained in the format

COB ID	RTR	DLC	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7

You will find this structure in most CAN analysis tools, and the data range Byte 0 to Byte 7 can be adapted to the respective CANopen message frame. RTR denotes the bit for a Remote-Transmit-Request. The DLC field consists of 4 bits and denotes the number of data bytes (DLC = Data Length Code).

3.4.2 NMT message frame

Two data bytes are transmitted per message frame. Data byte 0 contains the command specifier CS, Data byte 1 contains the device address. If the address 0 is entered, then all nodes will be addressed with the appropriate command (broadcast).

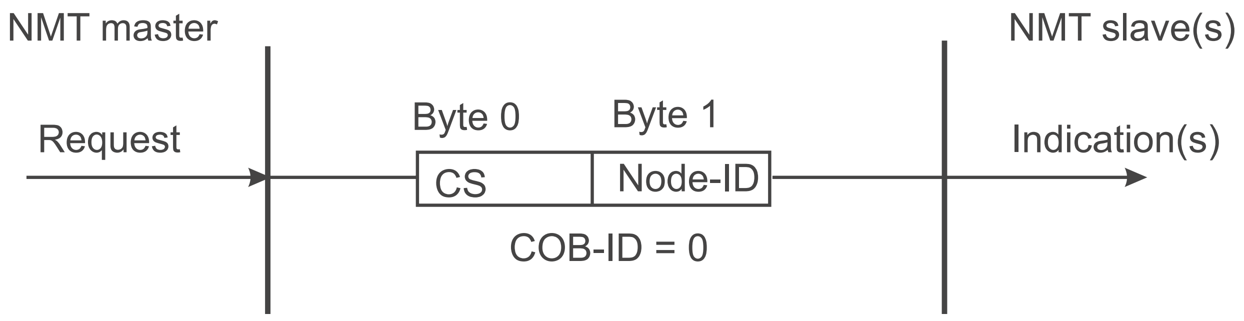


Figure 32: Communication relationship NMT

CS	Description	Effect
1	Start_Remote_Node	Starts normal operation
2	Stop_Remote_Node	Deactivates PDO and SDO communication
128	Enter_Pre-Operational_State	Transition to configuration mode
129	Reset_Node	Controlled reset of entire object directory to default values
130	Reset_Communication	Reset of the communication section of the object directory to default values

A message frame bringing Node 16 into configuration mode has the following construction:

COB ID	RTR	DLC	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
00h	0	2h	80h	10h						

These message frames are unconfirmed, i.e. no NMT slave acknowledges the correctly received message to the NMT master.

3.4.3 SDO message frame

3.4.3.1 Structure of an SDO message frame

The COB ID of the request SDO (request from the Master) results from 600h + address, and for response SDOs (Slave’s response) from 580h + address. The data field of the CAN data message frame (8 bytes) for an SDO is organized into three parts, a command specifier CS (1 byte), a multiplexor M (3 bytes) and the user data area D0 - D4 (4 bytes).

COB ID	RTR	DLC	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
600h/ 580h + Address	0	8h	CS	M			D0	D1	D2	D3

The multiplexor M consists of the 16-bit index of an object and the related 8-bit-wide subindex. With segmented message frames (not supported by the CANopen Slave option module), the user data range is extended by the three bytes of the multiplexor, so that 7 bytes of user data can be transferred. The command specifier CS classifies the various SDO types.

3.4.3.2 Types of SDO transfers

The CANopen Slave option module supports Expedited Transfer as per DS 301. Objects can be written or read, with their data comprised of a maximum of 4 bytes. Only two message frames are required, a request and a response.

3.4.3.3 Writing objects via SDO

A CANopen Master sends (Client) a write request to the CANopen Slave option module (Server). This carries out the request and acknowledges this with the response.

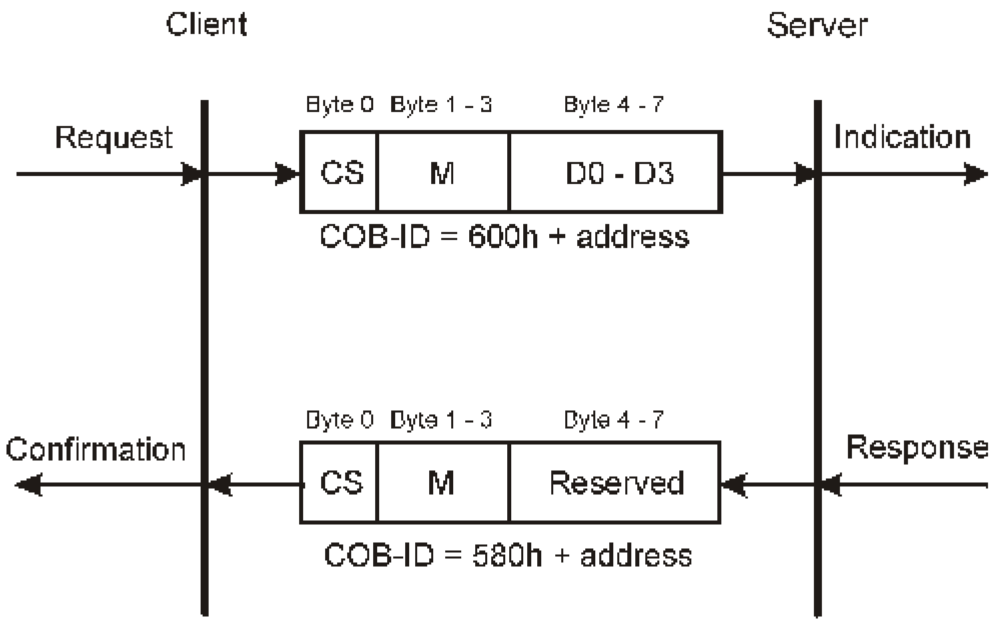


Figure 33: Writing the SDO communication relationship

The command specifier CS for the request will depend on the length of the user data. D0 is the LSB, D3 the MSB.

Data length in D0 - D3	Command specifier CS
1 Byte	2Fh
2 Byte	2Bh
4 Byte	23h

The command specifier CS for the response is 60h, the multiplexor M is identical to that of the request, the data field without meaning (reserved).

3.4.3.4 Reading objects via SDO

A CANopen Master sends (Client) a read request to the CANopen Slave option module (Server). This carries out the request and acknowledges this with the response, which also contains the data.

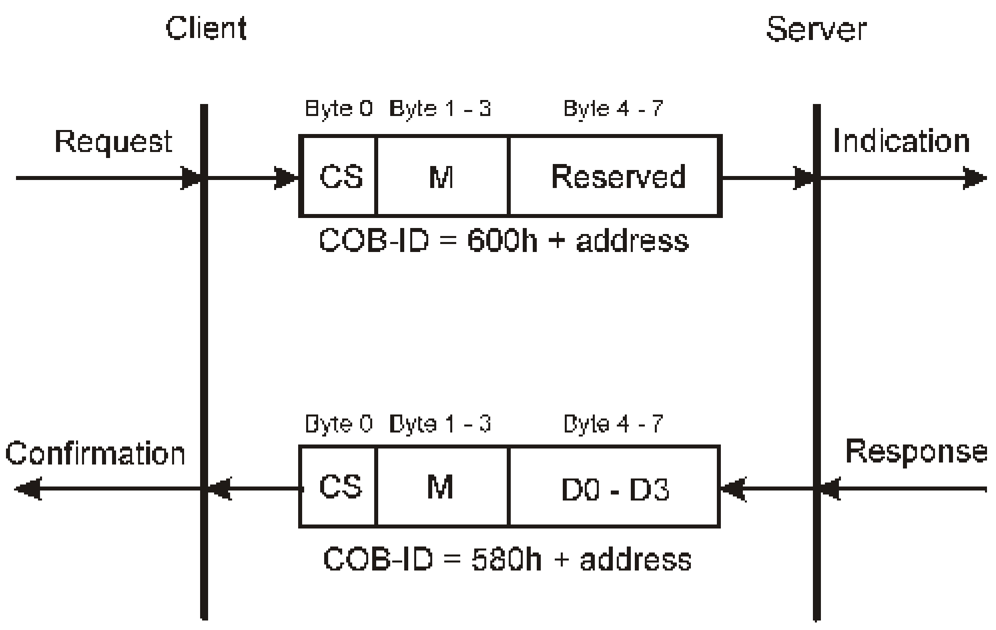


Figure 34: Reading the SDO communication relationship

An SDO server (master) sends a read request to the slave. This slave carries out the request and sends the required data in the response message frame (response). The command specifier CS for the request is always 40h. The command specifier CS for the response will depend on the length of the user data. D0 is the LSB, D3 the MSB.

Data length in D0 - D3	Command specifier CS
1 Byte	4Fh
2 Byte	4Bh
4 Byte	43h

The request and response multiplexors agree.

3.4.4 PDO message frame

A PDO producer sends the process data to the CANopen network in a PDO. Depending on the configuration, one or more PDO consumers evaluate the PDO.

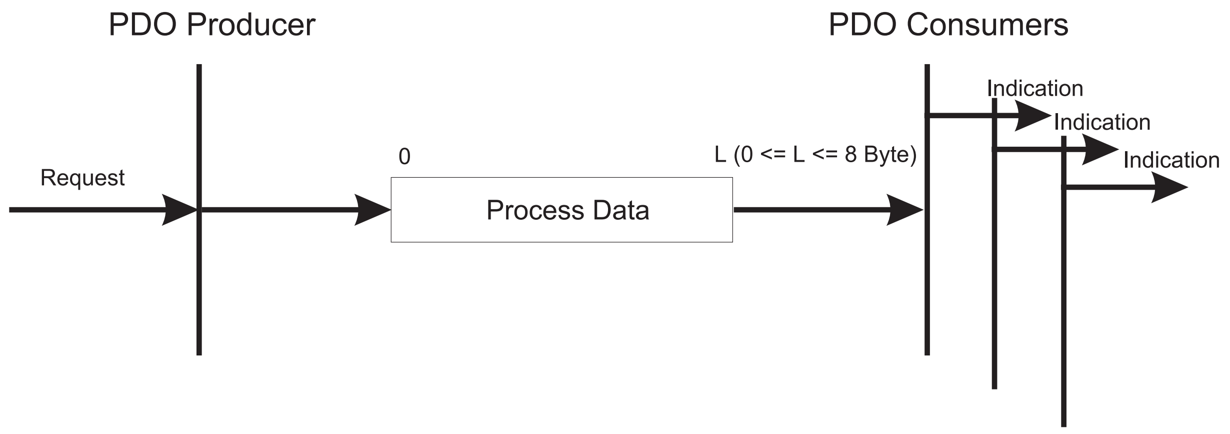


Figure 35: PDO communication relationship

All 8 bytes of the data range in the CAN data message frame are available for PDO message frames. The contents result from the settings of the mapping objects. Values from 181h to 57Fh result for the COB ID, provided that the COB ID has been assigned according to the predefined identifier assignment of the CiA. In addition, the COB ID can also be freely set.

COB ID	RTR	DLC	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
181h - 57Fh or any	0	0h - 8h	D0	D1	D2	D3	D4	D5	D6	D7

With remote requests of PDO message frames, the COB ID of PDO request and PDO response has the same value. The RTR bit (constituent of the DLC) is set in the PDO request. The data bytes have no contents, therefore the DLC field is 0.

COB ID	RTR	DLC	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
181h - 57Fh or any	1	0h	D0	D1	D2	D3	D4	D5	D6	D7

3.4.5 SYNC message frame

A SYNC producer sends the SYNC message frame to the CANopen network. Depending on the configuration, one or more SYNC consumers evaluate the message frame.

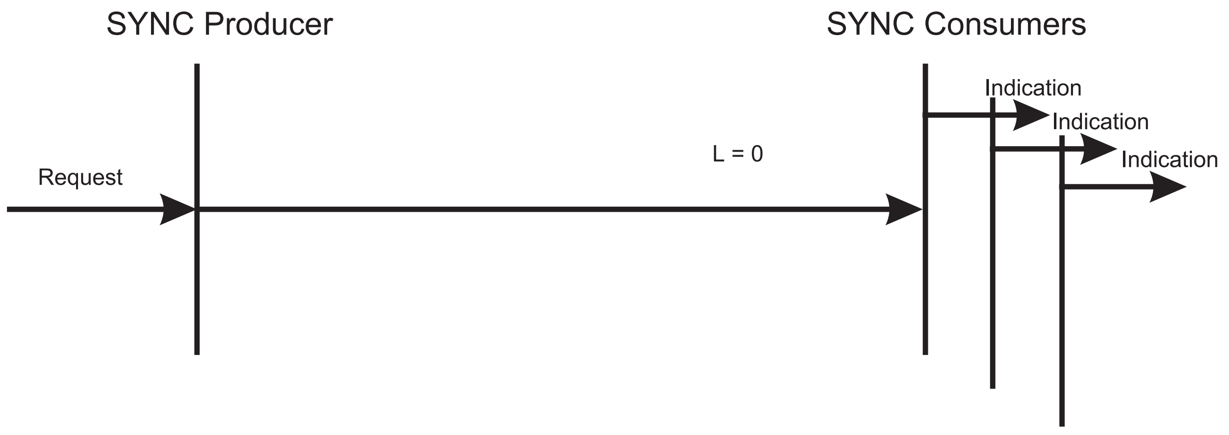


Figure 36: SYNC communication relationship

The SYNC message frame does not contain any data. The COB ID is 80h according to the predefined identifier assignment of the CiA. In addition, the COB ID can also be freely set.

COB ID	RTR	DLC	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
80h or any	0h	0h	-	-	-	-	-	-	-	-

3.4.6 Node Guarding message frame

The master interrogates the slaves at certain intervals by means of remote frames.

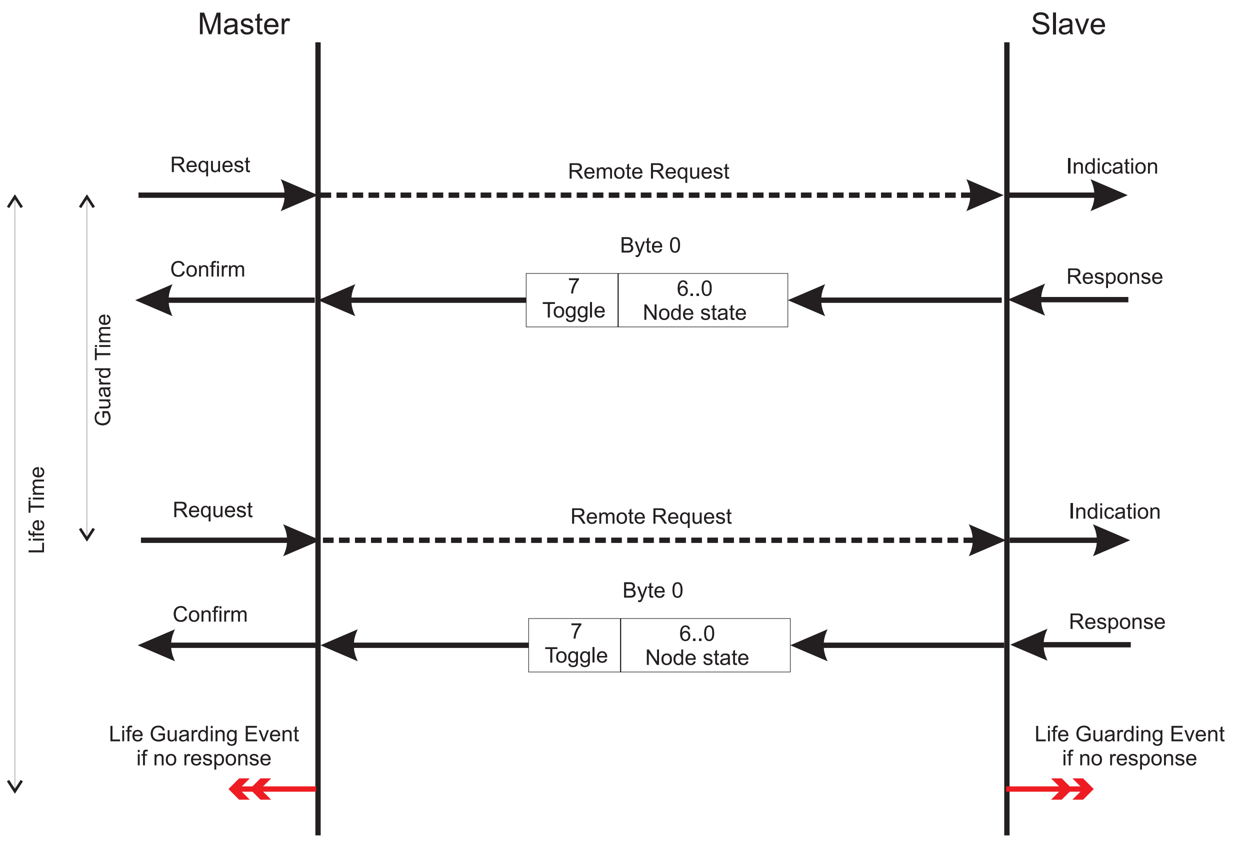


Figure 37: Node Guarding communication relationship

The COB ID results from 700h + address. The data field of the CAN data message frame consists of one byte, Byte 0, in the response message frame. Byte 0 is divided into the toggle bit (Bit 7) and the node state (Bit 6..0) (node state = NMT state).

COB ID	RTR	DLC	Byte 0
700h + Address	0	1h	Toggle bit + Node State

The remote request does not contain any data. The RTR bit is set.

COB ID	RTR	DLC	Byte 0
700h + Address	1	0h	-

3.4.7 Heartbeat message frame

A heartbeat producer sends a heartbeat message frame at defined intervals. No response is expected.

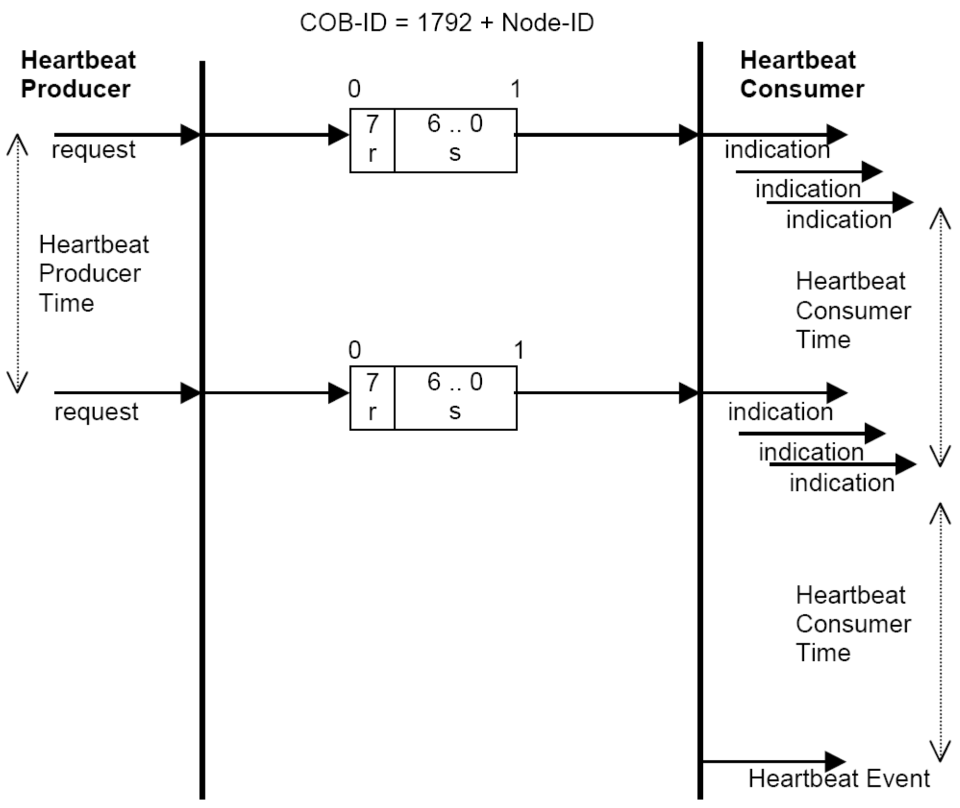


Figure 38: Heartbeat communication relationship

The COB ID results from 700h + address. The data field of the CAN data message frame consists of one byte, Byte 0. Byte 0 contains a reserved bit (Bit 7 = 0) and the NMT state (Bit 6..0) of the heartbeat producer.

COB ID	**RTR**	**DLC**	**Byte 0**
700h + Address	0	1	NMT state

3.4.8 Emergency message frame

An emergency producer sends the emergency message frame to the CANopen network. Depending on the configuration, one or more Emergency consumers evaluate the message frame.

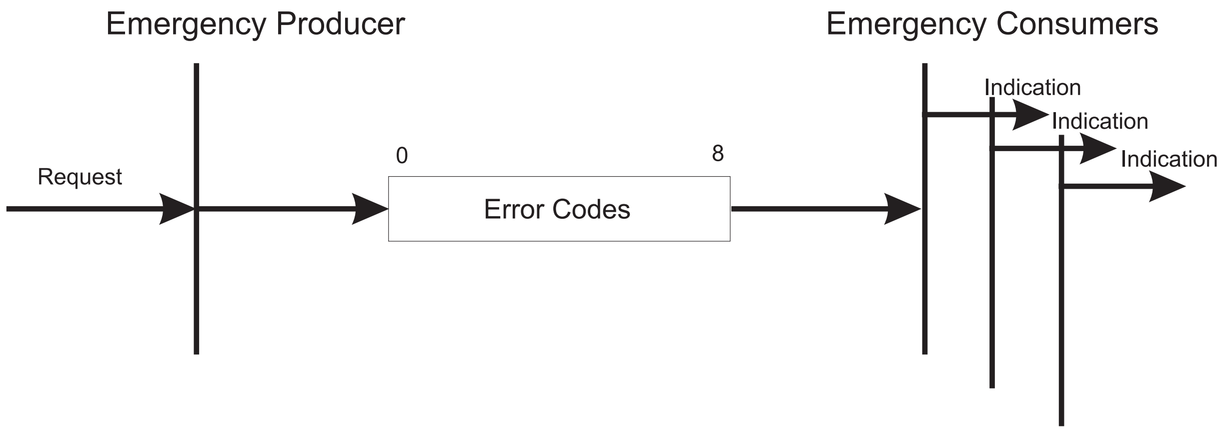


Figure 39: Emergency communication relationship

The COB ID results from 80h + address. All 8 bytes of the data range in the CAN data message frame are available for emergency message frames. The 8 bytes are organized as follows:

COB ID	RTR	DLC	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
80h + Address	0h	8h	Emergency error code		Error reg.	Manufacturer-specific				

APPENDIX A - ABBREVIATIONS

CAL	CAN Application Layer
CAN	Controller Area Network
CiA	CAN in Automation e. V., www.can-cia.de
COB	Communication object
COB ID	Communication Object Identifier
CS	Command specifier
CSMA/CA	Carrier Sense Multiple Access / Collision Avoidance
CSMA/CD	Carrier Sense Multiple Access / Collision Detection
DCF	Device Configuration File
DIN	Deutsches Institut für Normung e.V. (German Standards Institute)
DLC	Data Length Code
DS	Draft Standard
DSP	Draft Standard Proposal
EDS	Electronic Data Sheet
EMCY	Emergency
EN	European standard
FB	Function block, (IEC 61131-3)
IEC	International Electroctechnical Commission, www.iec.ch
ISO	International Standard Organisation, www.iso.ch
LED	Light Emitting Diode
LSB	Last Significant Bit
MSB	Most Significant Bit
NMT	Network Management
Node ID	Node Identifier, Address, Node Address, Network Node Address, CANopen Slave Address
PDO	Process Data Object
PLC	Process Loop Controller
POU	Program Organisation Unit, IEC 61131-3
PROPROG wt II	Tool for programming the b maXX PLC
RTR	Remote Transmission Request
RxPDO	Receive-PDO
SDO	Service Data Object
PLC	Process Loop Controller
TxPDO	Transmit-PDO
WinBASS II	Tool for parameterization of the b maXX controller
WWW	World Wide Web
16#	Prefix for hexadecimal numbers

APPENDIX B - TABLES

The following tables list the objects supported by the CANopen Slave option module.

B.1 Objects according to DS 301

In this section you will find all objects of the communication-specific area of the object directory that are supported by the Baumüller CANopen option module in accordance with DS301.

Overview of the objects:

Index	Object content / meaning
1000h	Device type: Supported device profile and additional information
1001h	Error register
1002h	
1003h	Error memory contains an error list of the unit
1004h	
1005h	SYNC message frame COB ID
1006h	SYNC interval
1007h	
1008h	
1009h	
100Ah	
100Bh	
100Ch	Guarding Time
100Dh	Lifetime Factor
100Eh	
100Fh	
1010h	Store parameter

B.1 Objects according to DS 301

Index	Object content / meaning
1011h	Load default parameter
1014h	
1400h - 15FFh	Receive-PDO (RxPDO) communication parameter
1600h - 17FFh	Receive-PDO (RxPDO) mapping parameter
1800h - 19FFh	Send-PDO (TxPDO) communication parameter
1A00h - 1BFFh	Send-PDO (TxPDO) mapping parameter

1000h - Device Type

Displays the supported unit profile and additional information about the unit.

Index	Subindex	Name	Data type	Access	Category	Mappable
1000h	00h	Device Type	Unsigned32	ro	Mandatory	No

This object is read-only and contains information on the related device (IEC 61131-3 Programmable Device according to DS 405).

In subindex 00h, 00000195h (405dec) is set as default for the CANopen Slave option module for b maXX PLC.

1001h - Error register

Contains a bit strip with pending device errors

Index	Subindex	Name	Data type	Access	Category	Mappable
1001h	00h	Error Register	Unsigned8	ro	Mandatory	Optional

This object is read-only. Object 1001h contains an error bit string with the following meaning:

Bit	Meaning
0	Error occurred
1	Not used
2	Not used
3	Not used
4	CAN communication error
5	Not used
6	Not used
7	Manufacturer-specific error

This error bit strip is OR-linked with 80h (manufacturer-specific error) when the FB CANop405_EMCY_SEND_SL is activated.

1003h - Predefined Error Field

Contains an error list of the device in descending time order.

Index	Subindex	Name	Data type	Access	Category	Mappable
1003h	-	Pre-defined error field	Array	-	Optional	-
	0h	Number of errors	Unsigned8	rw	Mandatory	No
	1h - FEh	Standard error field	Unsigned32	ro	Optional	No

This object can only be written to Subindex 00h. This contains an error history of 15 errors maximum, in which the last-occurred error is in Subindex 01h.

Subindex 00h of this object holds the number of errors registered. By writing "00h" to Subindex 00h, the list will be cleared.

The error code results from the definition of the *Emergency Error Code* in the emergency message frame:

Error Code	Meaning
00xxh	Error Reset or No Error
10xxh	Generic error
20xxh	Current
21xxh	Current, device input side
22xxh	Current inside the device
23xxh	Current, device output side
30xxh	Voltage
31xxh	Mains Voltage
32xxh	Voltage inside the device
33xxh	Output Voltage
40xxh	Temperature
41xxh	Ambient temperature
42xxh	Device Temperature
50xxh	Device Hardware
60xxh	Device Software
61xxh	Internal Software
62xxh	User Software
63xxh	Data Set
70xxh	Additional Modules
80xxh	Monitoring
81xxh	Communication
8110h	CAN Overrun (Objects lost)
8120h	CAN in Error Passive Mode

Error Code	Meaning
8130h	Life Guard Error or Heartbeat Error
8140h	recovered from bus off
8150h	Transmit COB ID
82xxh	Protocol Error
8210h	PDO not processed due to length error
8220h	PDO length exceeded
90xxh	External Error
F0xxh	Additional Functions
FFxxh	Device-specific

xx = Not defined

1004h - Number of PDOs supported

Index	Subindex	Name	Data type	Access	Category	Mappable
1004h	00h	Number of PDOs supported	Unsigned32			
	01h	Number of errors	Unsigned32			
	02h	Standard error field	Unsigned32			

This object is read-only. Subindex 00h contains the total number of supported PDOs, where the number of Receive-PDOs is in the high byte and the number of Transmit-PDOs in the low byte. Subindex 01h contains the possible number of synchronized PDOs, sub-index 02h the possible number of asynchronous PDOs.

The values entered mean that 4 Receive-PDOs and 4 Transmit-PDOs are available, whereby each PDO can be defined as synchronized or asynchronous.

1005h - COB ID SYNC-Message

Contains the COB ID of the SYNC message frame and indicates whether the unit generates the SYNC message frame.

Index	Subindex	Name	Data type	Access	Category	Mappable
1005h	00h	COB ID SYNC message	Unsigned32	rw	Conditional	No

This object contains information on the sync behavior of the slave. The MSB in the high word signifies that the Slave does not generate the SYNC message frame, but only receives it. The lower 11 bits in the low word specify the identifier of the SYNC message frame (80h).

1006h - Communication Cycle Period

Length of the SYNC interval in µs.

Index	Subindex	Name	Data type	Access	Category	Mappable
1006h	00h	Communication cycle period	Unsigned32	rw	Conditional	No

Should the SYNC message frame be activated, the sync interval must be set to the SYNC message frame period (1000, 2000, 4000 or 8000 µs). The set time affects the SYNC monitoring (see Chapter ▹Synchronization of data exchange ◃ on page 67).

1007h - Synchronous Window Length

Index	Subindex	Name	Data type	Access	Category	Mappable
1007h	00h	Synchronous window length				

This object is not evaluated internally.

1008h - Manufacturer Device Name

Index	Subindex	Name	Data type	Access	Category	Mappable
1008h	00h	Manufacturer device name	VString			

This object is read-only. It contains the character string: "b maXX PLC".

1009h - Manufacturer Hardware Version

Index	Subindex	Name	Data type	Access	Category	Mappable
1009h	00h	Manufacturer hardware version	VString			

This object is read-only. It contains the current hardware version of the option module, e.g. the character string: "HV01.00".

100Ah - Manufacturer Software Version

Index	Subindex	Name	Data type	Access	Category	Mappable
100Ah	00h	Manufacturer software version	VString			

This object is read-only. It contains the current software version of the option module, e.g. the character string: "SV01.00".

100Bh - Node ID

Index	Subindex	Name	Data type	Access	Category	Mappable
100Bh	00h	Node ID	Unsigned32			

This object is read-only. It contains the currently set Node ID (network node address). Only the low byte is valid.

100Ch - Guarding Time

The distance between two guarding message frames in ms.

Index	Subindex	Name	Data type	Access	Category	Mappable
100Ch	00h	Guard time	Unsigned16	rw	Conditional	No

In this object, the time base for node guarding is set in milliseconds. By writing the value "0", node guarding will be deactivated.

100Dh - Life Time Factor

Multiplied by the guarding time (object 100Ch) yields the life time in ms.

Index	Subindex	Name	Data type	Access	Category	Mappable
100Dh	00h	Life time factor	Unsigned8	rw	Conditional	No

The value of this object is multiplied by object 100Ch and from this the Node Guarding period results.

100Eh - COB ID Guarding Protocol

Index	Subindex	Name	Data type	Access	Category	Mappable
100Eh	00h	COB ID Guarding Protocol	Unsigned32			

This object contains the identifier of the Node Guarding Protocol.

100Fh - Number of SDOs supported

Index	Subindex	Name	Data type	Access	Category	Mappable
100Fh	00h	Number of SDOs supported	Unsigned32			

This object is read-only. It contains the number of supported SDOs. The high word holds the number of client SDOs supported by the device. The low word contains the number of server SDOs of the device.

1010h - Store parameters

Makes it possible to store parameters in non-volatile memory.

Index	Subindex	Name	Data type	Access	Category	Mappable
1010h	00h	Store parameters	Unsigned8			
	01h	Save all parameters (not supported)	Unsigned32			
	02h	Save communication parameters	Unsigned32			

This object supports the saving of parameters in non-volatile memory. Subindex 00h is read-only and contains the largest subindex to be supported (here, 2). Subindex 01h is currently not supported. The 1h in subindex 02h indicates that saving is supported, here in particular saving mapping and communication parameters. Saving, only possible in the PRE-OPERATIONAL state, is initiated with the value 65766173h as U32 or with the value "save" as string and lasts a few hundred milliseconds. The values last set for the mapping and communication parameters will be saved.

1011h - Restore Parameters

Allows you to activate default parameters.

Index	Subindex	Name	Data type	Access	Category	Mappable
1011h	00h	Restore parameters	Unsigned8	ro	Mandatory	No
	01h	Restore all parameters (not supported)	Unsigned32			
	02h	Restore default commu-nication parameters	Unsigned32			

The communication parameters are set to default values for this object. Subindex 00h is read-only and contains the largest subindex to be supported. Subindex 01h is currently not supported. With the value 1h in subindex 02h, the mapping and communication parameter default values will be set. This occurs only in the PRE-OPERATIONAL state by entering the value 64616F6Ch (U32) or with the input "load" as string parameter.

1014h - COB ID emergency

Index	Subindex	Name	Data type	Access	Category	Mappable
1014h	00h	COB ID emergency	Unsigned32			

This object contains the identifier of the EMCY message frame.

1400h-15FFh - Receive-PDO (RxPDO) Communication Parameter

Objects 1400h to 15FFh determine the properties for transferring RxPDO 1 to RxPDO 512. Object 1400h writes to RxPDO 1 and – assuming the unit supports it – object 15FFh writes to RxPDO 512.

Index	Subindex	Name	Data type	Access	Category	Mappable
1400h - 15FFh	-	RxPDO parameter	CiA defined	-	Conditional	-
	0h	Largest sub-index sup-ported	Unsigned8	rw	Mandatory	No
	1h	COB ID Receive RxPDO	Unsigned32	ro / rw	Mandatory	No
	2h	Transmission type RxPDO	Unsigned8	ro / rw	Mandatory	No

Subindex 0 indicates the number of the last valid subindex. The individual bits of the 32-bit value of **subindex 1** have the following meanings:

Subindex 1, bit	Value / meaning
31 (MSB)	0: PDO exists and is valid 1: PDO does not exist or is not valid According to the predefined identifier assignment, only 4 PDOs are possible for writing and 4 PDOs for reading. If a device supports more than these four PDOs, the COB IDs for the additional PDOs must be assigned themselves (bits 10 - 0 in this subindex). For this reason, bit 31 of the additional PDOs usually has a value of 1 and users do not set it to 0 until they have assigned a COB ID.
30	0: The PDO can be requested by means of a remote request 1: The PDO cannot be requested by means of a remote request
29	0: 11-bit CAN ID 1: 29-bit CAN ID (is not supported by the CANopen Slave option module)
28 - 11	Bits 28 - 11 with 29-bit CAN ID (is not supported by the CANopen Slave option module)
10 - 0 (LSB)	COB ID of the PDO. In the default settings of a device, the first four PDOs for reading have the COB IDs after the predefined identifier assignment.

Subindex 2 determines the PDO's transfer type:

Subindex 2, value	Type	Meaning
0	Synchronized	The system applies the PDO with an appropriate COB ID that was received before the last SYNC message frame
1 - 240	Synchronized	The system applies the PDO with an appropriate COB ID that was received before the last SYNC message frame
252	Remote	*Not possible*
253	Remote	*Not possible*
254	Asynchro-nous	The system applies each PDO with an appropriate COB ID when it is received
255	Asynchro-nous	The system applies each PDO with an appropriate COB ID when it is received

1600h-17FFh - Receive-PDO (RxPDO) Mapping Parameter

Objects 1600h to 17FFh determine the mapped objects of RxPDO 1 to RxPDO 512. Object 1600h writes to RxPDO 1 and – assuming the unit supports it – object 17FFh writes to RxPDO 512.

Index	Subindex	Name	Data type	Access	Category	Mappable
1600h - 17FFh	-	RxPDO mapping	CiA defined	-	Conditional	-
	0h	Number of mapped objects	Unsigned8	rw	Mandatory	No
	1h - 40h	Object to be mapped	Unsigned32	rw	Conditional	No

Subindex 0 indicates the number of mapped objects in the PDO. As soon as the mapping of a PDO is to be changed, you must first set the number of mapped objects to zero. After this, you can set the mapping and finally enter the number of mapped objects in subindex 0. In subindex 1h to 40h, the mapping information of the objects in the PDO is stated. Subindex 1h contains the information for the first mapped object; subindex 2h contains the information for the second one, etc. A maximum of 64 (= 40h) objects can be mapped; in this connection, you cannot exceed the maximum data range of 8 bytes of the PDO. The contents of the 32-bit values of subindexes 1h - 40h have the following meanings:

Byte: MSB LSB

Index of the mapped object (16-bit)	Subindex of the mapped object (8-bit)	Number of bits of the mapped object (8-bit)

The system maps the objects successively in one PDO.

1800h-19FFh - Send-PDO (TxPDO) Communications Parameter

Objects 1800h to 19FFh determine the properties for transferring TxPDO 1 to TxPDO 512. Object 1800h writes to TxPDO 1 and – assuming the unit supports it – object 19FFh writes to TxPDO 512.

Index	Subindex	Name	Data type	Access	Category	Mappable
1800h - 19FFh	-	TxPDO parameter	CiA defined	-	Conditional	-
	0h	Largest sub-index supported	Unsigned8	rw	Mandatory	No
	1h	COB ID Receive TxPDO	Unsigned32	ro / rw	Mandatory	No
	2h	Transmission type TxPDO	Unsigned8	ro / rw	Mandatory	No
	3h	Inhibit time TxPDO	Unsigned16	rw	Optional	No
	4h	Reserved	-	-	-	-
	5h	Event Timer TxPDO	Unsigned16	rw	Optional	No

Subindex 0 indicates the number of the last valid subindex. The individual bits of the 32-bit value of **subindex 1** have the following meanings:

Subindex 1, bit	**Value / meaning**
31 (MSB)	0: PDO exists and is valid 1: PDO does not exist or is not valid According to the predefined identifier assignment, only 4 PDOs are possible for writing and 4 PDOs for reading. If a device supports more than these four PDOs, the COB IDs for the additional PDOs must be assigned themselves (bits 10 - 0 in this subindex). For this reason, bit 31 of the additional PDOs usually has a value of 1 and users do not set it to 0 until they have assigned a COB ID.
30	0: The PDO can be requested by means of a remote request 1: The PDO cannot be requested by means of a remote request
29	0: 11-bit CAN ID 1: 29-bit CAN ID (is not supported by the CANopen Slave option module)
28 - 11	Bits 28 - 11 with 29-bit CAN ID (is not supported by the CANopen Slave option module)
10 - 0 (LSB)	COB ID of the PDO. In the default settings of a device, the first four PDOs for writing have the COB IDs after the predefined identifier assignment.

Subindex 2 determines the PDO's transfer type:

Subindex 2, value	**Type**	**Meaning**
0	Synchronized	The system sends after each received SYNC message frame, but only if the transferred data are changed
1 - 240	Synchronized	The system sends after receiving the set number (1 - 240) of SYNC message frames
252	Remote	On receiving a SYNC message frame, the system updates the PDO. The system does not send until it receives a remote request.
253	Remote	The system updates and sends after receiving a remote request.
254	Asynchronous	The system sends on a manufacturer-specific basis. Note: With this type, most devices transfer the PDO on a time (when a timer runs down). The time is set under subindex 5.
255	Asynchronous	The system sends in dependence on the supported device profile. Note: With this type, most devices transfer the PDO as soon as one of the mapped values has changed.

Using **subindex 3,** you set a PDO inhibit time in multiples of 100 µs. This does not extend the response time at the relative first value change; rather, it is active on the changes that directly follow this one. The inhibit time describes the minimum period of time that the system must wait between sending two of the same message frames.

In **subindex 5,** you set the time in ms for event-driven PDO transfer assuming that a device supports event-driven PDO transfer (subindex 2 = 254 or 255 depending on the device profile).

1A00h-1BFFh - Send-PDO (TxPDO) Mapping Parameter

Objects 1A00h to 1BFFh determine the mapped objects of TxPDO 1 to TxPDO 512. Object 1A00h writes to TxPDO 1 and – assuming the unit supports it – object 1AFFh writes to TxPDO 512.

Index	Subindex	Name	Data type	Access	Category	Mappable
1A00h - 1BFFh	-	TxPDO mapping	CiA defined	-	Conditional	-
	0h	Number of mapped objects	Unsigned8	rw	Mandatory	No
	1h - 40h	Object to be mapped	Unsigned32	rw	Conditional	No

Subindex 0 indicates the number of mapped objects in the PDO. As soon as the mapping of a PDO is to be changed, you must first set the number of mapped objects to zero. After this, you can set the mapping and finally enter the number of mapped objects in subindex 0. The mapping information of the objects in the PDO is stated in **subindex 1h to 40h**. Subindex 1h contains the information for the first mapped object; subindex 2h contains the information for the second one, etc. A maximum of 64 (= 40h) objects can be mapped; in this connection, you cannot exceed the maximum data range of 8 bytes of the PDO. The contents of the 32-bit values of **subindexes 1h to 40h** have the following meanings:

Byte:	MSB		LSB
	Index of the mapped object (16-bit)	Subindex of the mapped object (8-bit)	Number of bits of the mapped object (8-bit)

The system maps the objects successively in one PDO.

B.2 Objects according to DS 405

The CANopen Slave option module for b maXX PLC supports the following objects from the DS 405:

1000h - Device Type

Displays the supported unit profile and additional information about the unit.

Index	Subindex	Name	Data type	Access	Category	Mappable
1000h	00h	Device Type	Unsigned32	ro	Mandatory	No

This object is read-only and contains information on the related device (IEC 61131-3 Programmable Device according to DS 405).

In subindex 00h, 00000195h (405dec) is set as default for the CANopen Slave option module for b maXX PLC.

A00h to A6FF

Index	Subindex	Name	Data type	Access	Category	Mappable
A000h	01h	1st Dynamic Integer8	Integer8	ro	Optional	Yes
...						
A03Fh	64h	64th Dynamic Integer8	Integer8	ro	Optional	Yes

Index	Subindex	Name	Data type	Access	Category	Mappable
A040h	01h	1st Dynamic Unsigned8	Unsigned8	ro	Optional	Yes
...						
A07Fh	64h	64th Dynamic Unsigned8	Unsigned8	ro	Optional	Yes

Index	Subindex	Name	Data type	Access	Category	Mappable
A0C0h	01h	1st Dynamic Integer16	Integer16	ro	Optional	Yes
...						
A0FFh	64h	64th Dynamic Integer16	Integer16	ro	Optional	Yes

Index	Subindex	Name	Data type	Access	Category	Mappable
A100h	01h	1st Dynamic Unsigned16	Unsigned16	ro	Optional	Yes
...						
A13Fh	64h	64th Dynamic Unsigned16	Unsigned16	ro	Optional	Yes

Index	Subindex	Name	Data type	Access	Category	Mappable
A1C0h	01h	1st Dynamic Integer32	Integer32	ro	Optional	Yes
...						
A1FFh	64h	64th Dynamic Integer32	Integer32	ro	Optional	Yes

Index	Subindex	Name	Data type	Access	Category	Mappable
A200h	01h	1st Dynamic Unsigned32	Unsigned32	ro	Optional	Yes
...						
A23Fh	64h	64th Dynamic Unsigned32	Unsigned32	ro	Optional	Yes

Index	Subindex	Name	Data type	Access	Category	Mappable
A240h	01h	1st Dynamic Float(32)	Float(32) (=Real)	ro	Optional	Yes
...						
A27Fh	64h	64th Dynamic Float(32)	Float(32)	ro	Optional	Yes

Index	Subindex	Name	Data type	Access	Category	Mappable
A480h	01h	1st Dynamic Integer8	Integer8	rw	Optional	Yes
...						
A4BFh	64h	64th Dynamic Integer8	Integer8	rw	Optional	Yes

Index	Subindex	Name	Data type	Access	Category	Mappable
A4C0h	01h	1st Dynamic Unsigned8	Unsigned8	rw	Optional	Yes
...						
A4FFh	64h	64th Dynamic Unsigned8	Unsigned8	rw	Optional	Yes

Objects according to DS 405

Index	Subindex	Name	Data type	Access	Category	Mappable
A540h	01h	1st Dynamic Integer16	Integer16	rw	Optional	Yes
...						
A57Fh	64h	64th Dynamic Integer16	Integer16	rw	Optional	Yes

Index	Subindex	Name	Data type	Access	Category	Mappable
A580h	01h	1st Dynamic Unsigned16	Unsigned16	rw	Optional	Yes
...						
A5BFh	64h	64th Dynamic Unsigned16	Unsigned16	rw	Optional	Yes

Index	Subindex	Name	Data type	Access	Category	Mappable
A640h	01h	1st Dynamic Integer32	Integer32	rw	Optional	Yes
...						
A67Fh	64h	64th Dynamic Integer32	Integer32	rw	Optional	Yes

Index	Subindex	Name	Data type	Access	Category	Mappable
A680h	01h	1st Dynamic Unsigned32	Unsigned32	rw	Optional	Yes
...						
A6BFh	64h	64th Dynamic Unsigned32	Unsigned32	rw	Optional	Yes

Index	Subindex	Name	Data type	Access	Category	Mappable
A6C0h	01h	1st Dynamic Float(32)	Float(32) (=Real)	rw	Optional	Yes
...						
A6FFh	64h	64th Dynamic Float(32)	Float(32)	rw	Optional	Yes

B.3 Example EDS file

The example EDS file is located on the CANopen CD "CANop405_PLC01_20bd03" (or higher).

In addition, you can generate the example EDS file with the CANopen Slave Configurator in the b maXX Drive-Configurator. In this connection, also see ▹Configuration of the CANopen Slave option module by using the CANopen Slave Configurator ◃ from page 31 onward.

- Open the PROPROG project CANopenSlave_Example.mwt (if necessary unzip CANopenSlave_Example.zwt from the CANopen CD "CANop405_PLC01_20bd03" (or higher)).
- Open the CANopen Slave Configurator from PROPROG wt II via Configuration_Tools → Drive_Configuration → "Option modules \ G: CAN-03-00-00" - context menu item "Configure".
- Save the EDS file. In this connection, also see ▹Generating an EDS file ◃ from page 42 onward.

B.4 Default mapping of the PDO

Default mappings are defined in the device profiles. Below, there is an overview of the default mapping values for device profiles DS 401 and DSP 402. The assignment that is shown below refers to the maximum configuration of a device. Depending on the hardware, fewer objects may be mapped. However, this does not mean that objects of a different type replace other ones in the mapping, e.g. analog inputs instead of digital inputs. If objects are dropped, the associated locations in the default mapping stay vacant. It is, of course, your choice whether you want to assign other objects to the vacant locations.

B.4.1 Default mapping for I/O modules according to DS 401

Object	Meaning	Mapping value index subindex length
1600h	First Receive-PDO.	
Subindex 0	Number of mapped objects	8
Subindex 1	Digital outputs 0 - 7	6200 01 08h
Subindex 2	Digital outputs 8 - 15	6200 02 08h
Subindex 3	Digital outputs 16 - 23	6200 03 08h
Subindex 4	Digital outputs 24 - 31	6200 04 08h
Subindex 5	Digital outputs 32 - 39	6200 05 08h
Subindex 6	Digital outputs 40 - 47	6200 06 08h
Subindex 7	Digital outputs 48 - 55	6200 07 08h
Subindex 8	Digital outputs 56 - 63	6200 08 08h

Default mapping of the PDO

Object	Meaning	Mapping value index subindex length
1601h	Second Receive-PDO.	
Subindex 0	Number of mapped objects	4
Subindex 1	Analog output 1 (16-bit resolution)	6411 01 10h
Subindex 2	Analog output 2 (16-bit resolution)	6411 02 10h
Subindex 3	Analog output 3 (16-bit resolution)	6411 03 10h
Subindex 4	Analog output 4 (16-bit resolution)	6411 04 10h
1602h	Third Receive-PDO.	
Subindex 0	Number of mapped objects	4
Subindex 1	Analog output 5 (16-bit resolution)	6411 05 10h
Subindex 2	Analog output 6 (16-bit resolution)	6411 06 10h
Subindex 3	Analog output 7 (16-bit resolution)	6411 07 10h
Subindex 4	Analog output 8 (16-bit resolution)	6411 08 10h
1603h	Fourth Receive-PDO.	
Subindex 0	Number of mapped objects	4
Subindex 1	Analog output 9 (16-bit resolution)	6411 09 10h
Subindex 2	Analog output 10 (16-bit resolution)	6411 0A 10
Subindex 3	Analog output 11 (16-bit resolution)	6411 0B 10h
Subindex 4	Analog output 12 (16-bit resolution)	6411 0C 10h

Object	Meaning	Mapping value index subindex length
1A00h	First Send-PDO.	
Subindex 0	Number of mapped objects	8
Subindex 1	Digital inputs 0 - 7	6000 01 08h
Subindex 2	Digital inputs 8 - 15	6000 02 08h
Subindex 3	Digital inputs 16 - 23	6000 03 08h
Subindex 4	Digital inputs 24 - 31	6000 04 08h
Subindex 5	Digital inputs 32 - 39	6000 05 08h
Subindex 6	Digital inputs 40 - 47	6000 06 08h
Subindex 7	Digital inputs 48 - 55	6000 07 08h
Subindex 8	Digital inputs 56 - 63	6000 08 08h
1A01h	Second Send-PDO.	
Subindex 0	Number of mapped objects	4

Object	Meaning	Mapping value index subindex length
Subindex 1	Analog input 1 (16-bit resolution)	6401 01 10h
Subindex 2	Analog input 2 (16-bit resolution)	6401 02 10h
Subindex 3	Analog input 3 (16-bit resolution)	6401 03 10h
Subindex 4	Analog input 4 (16-bit resolution)	6401 04 10h
1A02h	Third Send-PDO.	
Subindex 0	Number of mapped objects	4
Subindex 1	Analog input 5 (16-bit resolution)	6401 05 10h
Subindex 2	Analog input 6 (16-bit resolution)	6401 06 10h
Subindex 3	Analog input 7 (16-bit resolution)	6401 07 10h
Subindex 4	Analog input 8 (16-bit resolution)	6401 08 10h
1A03h	Fourth Send-PDO.	
Subindex 0	Number of mapped objects	4
Subindex 1	Analog input 9 (16-bit resolution)	6401 09 10h
Subindex 2	Analog input 10 (16-bit resolution)	6401 0A 10
Subindex 3	Analog input 11 (16-bit resolution)	6401 0B 10h
Subindex 4	Analog input 12 (16-bit resolution)	6401 0C 10h

B.4.2 Default mapping for drives according to DSP 402

Object	Meaning	Mapping value index subindex length
1600h	First Receive-PDO.	
Subindex 0	Number of mapped objects	1
Subindex 1	Control word (16-bit)	6040 00 10h
1601h	Second Receive-PDO.	
Subindex 0	Number of mapped objects	2
Subindex 1	Control word (16-bit)	6040 00 10h
Subindex 2	Operating mode (8-bit)	6060 00 08h
1602h	Third Receive-PDO.	
Subindex 0	Number of mapped objects	2
Subindex 1	Control word (16-bit)	6040 00 10h
Subindex 2	Target position (32-bit)	607A 00 20
1603h	Fourth Receive-PDO.	

Object	Meaning	Mapping value index subindex length
Subindex 0	Number of mapped objects	2
Subindex 1	Control word (16-bit)	6040 00 10h
Subindex 2	RPM speed reference value (32-bit)	60FF 00 20h
1604h	Fifth Receive-PDO.	
Subindex 0	Number of mapped objects	2
Subindex 1	Control word (16-bit)	6040 00 10h
Subindex 2	Target torque (32-bit)	6071 00 20h
1605h	Sixth Receive-PDO.	
Subindex 0	Number of mapped objects	2
Subindex 1	Control word (16-bit)	6040 00 10h
Subindex 2	RPM speed reference value (16-bit)	6042 00 10h
1606h	Seventh Receive-PDO.	
Subindex 0	Number of mapped objects	2
Subindex 1	Control word (16-bit)	6040 00 10h
Subindex 2	Digital outputs (32-bit)	6042 00 20h
1607h	Eighth Receive-PDO.	
Subindex 0	Number of mapped objects	2
Subindex 1	Control word (16-bit)	6040 00 10h
Subindex 2	Operating mode (8-bit)	6060 00 08h

Object	Meaning	Mapping value index subindex length
1A00h	First Send-PDO.	
Subindex 0	Number of mapped objects	1
Subindex 1	Status word (16-bit)	6041 00 10h
1A01h	Second Send-PDO.	
Subindex 0	Number of mapped objects	2
Subindex 1	Status word (16-bit)	6041 00 10h
Subindex 2	Operating mode display (8-bit)	6061 00 08h
1A02h	Third Send-PDO.	
Subindex 0	Number of mapped objects	2
Subindex 1	Status word (16-bit)	6041 00 10h
Subindex 2	Position actual value (32-bit)	6064 00 20h

Object	Meaning	Mapping value index subindex length
1A03h	Fourth Send-PDO.	
Subindex 0	Number of mapped objects	2
Subindex 1	Status word (16-bit)	6041 00 10h
Subindex 2	RPM speed actual value (32-bit)	606C 00 20h
1A04h	Fifth Send-PDO.	
Subindex 0	Number of mapped objects	2
Subindex 1	Status word (16-bit)	6041 00 10h
Subindex 2	Torque actual value (32-bit)	6077 00 20h
1A05h	Sixth Send-PDO	
Subindex 0	Number of mapped objects	2
Subindex 1	Status word (16-bit)	6041 00 10h
Subindex 2	RPM speed actual value (16-bit)	6044 00 10h
1A06h	Seventh Send-PDO.	
Subindex 0	Number of mapped objects	2
Subindex 1	Status word (16-bit)	6041 00 10h
Subindex 2	Digital inputs (32-bit)	60FD 00 20h

B.4.3 Default mapping for programmable controls according to DS 405

Object	Meaning	Mapping value index subindex length
1600h	First Receive-PDO.	
Subindex 0	Number of mapped objects	5
Subindex 1	Object A480 Sidx 01 (INTEGER8)	A480 01 08h
Subindex 2	Object A4C0 Sidx 01 (UNSIGNED8)	A4C0 01 08h
Subindex 3	Object A540 Sidx 01 (INTEGER16)	A540 01 10h
Subindex 4	Object A580 Sidx 01 (UNSIGNED16)	A580 01 10h
Subindex 5	Object A580 Sidx 02 (UNSIGNED16)	A580 02 10h
1601h	Second Receive-PDO.	
Subindex 0	Number of mapped objects	2
Subindex 1	Object A640 Sidx 01 (INTEGER32)	A640 01 20h
Subindex 2	Object A680 Sidx 01 (UNSIGNED32)	A680 01 20h
1602h	Third Receive-PDO.	

Object	Meaning	Mapping value index subindex length
Subindex 0	Number of mapped objects	4
Subindex 1	Object A480 Sidx 01 (INTEGER8)	A480 01 08h
Subindex 2	Object A4C0 Sidx 01 (UNSIGNED8)	A4C0 01 08h
Subindex 3	Object A540 Sidx 01 (INTEGER16)	A540 01 10h
Subindex 4	Object A680 Sidx 01 (UNSIGNED32)	A680 01 20h
1603h	Fourth Receive-PDO.	
Subindex 0	Number of mapped objects	3
Subindex 1	Object A540 Sidx 01 (INTEGER16)	A540 01 10h
Subindex 2	Object A580 Sidx 01 (UNSIGNED16)	A580 01 10h
Subindex 3	Object A680 Sidx 01 (UNSIGNED32)	A680 01 20h

Object	Meaning	Mapping value index subindex length
1A00h	First Send-PDO.	
Subindex 0	Number of mapped objects	5
Subindex 1	Object A000 Sidx 01 (INTEGER8)	A000 01 08h
Subindex 2	Object A040 Sidx 01 (UNSIGNED8)	A040 01 08h
Subindex 3	Object A0C0 Sidx 01 (INTEGER16)	A0C0 01 10h
Subindex 4	Object A100 Sidx 01 (UNSIGNED16)	A100 01 10h
Subindex 5	Object A100 Sidx 02 (UNSIGNED16)	A100 02 10h
1A01h	Second Send-PDO.	
Subindex 0	Number of mapped objects	2
Subindex 1	Object A1C0 Sidx 01 (INTEGER32)	A1C0 01 20h
Subindex 2	Object A200 Sidx 01 (UNSIGNED32)	A200 01 20h
1A02h	Third Send-PDO.	
Subindex 0	Number of mapped objects	4
Subindex 1	Object A000 Sidx 01 (INTEGER8)	A000 01 08h
Subindex 2	Object A040 Sidx 01 (UNSIGNED8)	A040 01 08h
Subindex 3	Object A0C0 Sidx 01 (INTEGER16)	A0C0 01 10h
Subindex 4	Object A200 Sidx 01 (UNSIGNED32)	A200 01 20h
1A03h	Fourth Send-PDO.	
Subindex 0	Number of mapped objects	3

Object	Meaning	Mapping value index subindex length
Subindex 1	Object A0C0 Sidx 01 (INTEGER16)	A0C0 01 10h
Subindex 2	Object A100 Sidx 01 (UNSIGNED16)	A100 01 10h
Subindex 3	Object A200 Sidx 01 (UNSIGNED32)	A200 01 20h

B.5 Default EDS file

The default EDS file can be found on the CANopen CD "CANop405_PLC01_20bd03" (or higher).

You can also generate the default EDS file with the CANopen Slave Configurator in the b maXX Drive-Configurator. In this connection, also see ▹Configuration of the CANopen Slave option module by using the CANopen Slave Configurator ◃ from page 31 onward.

- Open the CANopen Slave Configurator from PROPROG wt II via Configuration_Tools → Drive_Configuration → "Option modules \ G: CAN-03-00-00" - context menu item "Configure".
- Activate the checkbox "Default configuration".
- Save the EDS file. In this connection, also see ▹Generating an EDS file ◃ from page 42 onward.

B.5 Default EDS file

C

APPENDIX C - LIBRARY CANOP405_PLC01_20BD03

The library CANop405_PLC01_20bd03 (or higher) contains the function blocks (FBs) for the CANopen Master option module and the CANopen Slave option module for programming the CANopen on the b maXX PLC.

FB name	CANopen Master option module	CANopen Slave option module
CANop405_COB_ID	X	
CANop405_EMERGENCY	X	
CANop405_EMCY_SEND_SL		X
CANop405_GET_STATE_SL		X
CANop405_PDO_INIT	X	
CANop405_INIT	X	
CANop405_INIT_SL		X
CANop405_KERNEL_STATE	X	
CANop405_KERNEL_STATE_SL		X
CANop405_LIFE_GUARD_SL		X
CANop405_LOC_NODE_ID_SL		X
CANop405_NMT	X	
CANop405_NODE_GUARDING	X	
CANop405_PDO_READ	X	
CANop405_PDO_WRITE	X	
CANop405_SDO1_READ	X	
CANop405_SDO1_WRITE	X	
CANop405_SDO2_READ	X	
CANop405_SDO2_WRITE	X	

FB name	CANopen Master option module	CANopen Slave option module
CANop405_SDO3_READ	X	
CANop405_SDO3_WRITE	X	
CANop405_SDO4_READ	X	
CANop405_SDO4_WRITE	X	
CANop405_SDO5_READ	X	
CANop405_SDO5_WRITE	X	
CANop405_SDO6_READ	X	
CANop405_SDO6_WRITE	X	
CANop405_SDO7_READ	X	
CANop405_SDO7_WRITE	X	
CANop405_SDO8_READ	X	
CANop405_SDO8_WRITE	X	
CANop405_SYNC	X	

Index

be in motion

Baumüller Nürnberg GmbH   Ostendstraße 80-90   90482 Nuremberg Tel: +49(0)911-5432-0 Fax: +49(0)911-5432-130 **www.baumueller.de**