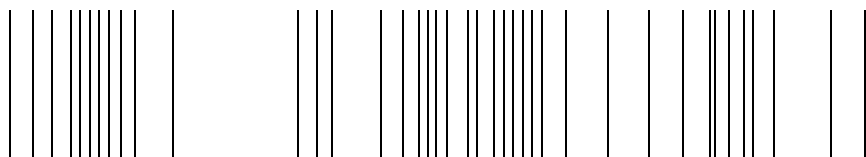


**be in motion be in motion**



**BMC-M-ETH-01/2,  
...-CAN-04**

**Ethernet + CANopen-Master  
for b maXX controller PLC**

**Application Manual**

**E**

5.04031.04



Title	Application Manual
Product	Ethernet + CANopen-Master for b maXX controller PLC BMC-M-ETH-01/2, ...-CAN-04
Last Revision:	January 29, 2007
Copyright	<p>Owners may make as many copies as they like of this Application Manual exclusively for their own internal use. You are not allowed to copy or duplicate even extracts from this Application Manual for any other purposes.</p> <p>You are not permitted to exploit or communicate the contents of this Application Manual.</p> <p>Any other designations or company logos used in this Application Manual may be trademarks whose use by third parties for their own purposes may affect the rights of the owner of the trademark.</p>
Binding nature	<p>This Application Manual is a part of the unit/machine. This Application Manual must always be available to operators and be legible. If the unit/machine is sold, the owner must pass on this Application Manual together with the unit/machine.</p> <p>After selling the unit/machine you must pass on this original and all the copies that you made to the purchaser. After disposing of the machine in any way, you must destroy this original and all the copies that you made.</p> <p>When you pass on this Application Manual, all earlier revisions of the corresponding Application Manuals are invalidated.</p> <p>Note that all the data/numbers/information that are quoted are <b>current values at the time of printing</b>. This information is <b>not legally binding</b> for dimensioning, calculation and costing.</p> <p>Within the scope of further-development of our products, Baumüller Nürnberg GmbH reserve the right to change the technical data and handling.</p> <p>We cannot guarantee this Application Manual is completely error-free unless this is expressly indicated in our General Conditions of Business and Delivery.</p>
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



<b>1</b>	<b>Introduction</b>	<b>5</b>
1.1	First Steps	5
1.2	Terms Used	6
1.3	Conditions	6
<b>2</b>	<b>Basic Safety Instructions</b>	<b>7</b>
2.1	Hazard information and instructions	7
2.1.1	Structure of hazard information	8
2.1.2	Hazard advisories that are used	9
2.2	Information signs	11
2.3	Legal information	11
2.4	Appropriate Use	11
2.5	Inappropriate Use	12
2.6	Protective equipment	12
2.7	Personnel training	13
2.8	Safety measures in normal operation	13
2.9	Responsibility and liability	13
2.9.1	Observing the hazard information and safety instructions	13
2.9.2	Danger arising from using this module	14
2.9.3	Warranty and Liability	14
<b>3</b>	<b>Ethernet</b>	<b>15</b>
3.1	General information on Ethernet and the Ethernet module	15
3.2	Basics of Ethernet Networks and TCP/IP	16
3.2.1	Transfer standards	16
3.2.2	Structure of Ethernet networks with a hub and switch	17
3.2.3	Ethernet address and MAC address	18
3.2.4	Ethernet data packet	18
3.2.5	Bridges and Repeaters	19
3.2.6	TCP/IP	19
3.2.7	Interaction between Ethernet and TCP/IP, ARP	28
3.2.8	Proxy	29
3.2.9	Application layer PROPROG wt II / OmegaOS	29
3.3	Configuring Ethernet TCP/IP networks	30
3.3.1	Overview	30
3.3.2	Configuring the Windows PC	30
3.3.3	Configuring the Ethernet module and checking settings	39
3.4	Setting a connection via Ethernet-TCP/IP in PROPROG wt II / ProProg wt III	47
<b>4</b>	<b>CANopen</b>	<b>49</b>
4.1	General information on CANopen and on use of CANopen master module	49
4.2	Basics of CAN and CANopen networks	51
4.2.1	Basics of CAN	51
4.2.2	Basics of CANopen	52
4.3	ProMaster, ProCANopen, ProProg wt III and Motion Control	57
4.3.1	Requirements	58
4.3.2	Steps to be carried out	58
4.3.3	Commissioning of the CANopen network	59
4.3.4	Creating a IEC project with ProProg wt III	59
4.3.5	Creation of a machine configuration in ProMaster	61
4.3.6	Configuring CANopen communication with ProCANopen	65
4.3.7	Configuring PLC	82
4.3.8	Downloading data to CANopen master and to b maXX controller PLC	95
4.3.9	Programming IEC project	96
4.4	Programming data exchange with PROPROG wt II and CANop405_PLC01_20bd03 li-	



# TABLE OF CONTENTS

---

	brary . . . . .	101
4.4.1	Overview . . . . .	101
4.4.2	Steps to be carried out. . . . .	101
4.4.3	Commissioning CANopen network . . . . .	102
4.4.4	Creation of a project and integration of the CANop405_PLC01_20bd03 library . . . . .	102
4.4.5	Creating a global variable for data exchange . . . . .	103
4.4.6	Initializing CANopen master module . . . . .	104
4.4.7	Starting individual network nodes - NMT . . . . .	106
4.4.8	Service data exchange with SDO . . . . .	108
4.4.9	Process data exchange with PDOs . . . . .	114
4.4.10	Synchronizing data exchange . . . . .	140
4.4.11	Monitoring network nodes with Node Guarding . . . . .	145
4.4.12	Receiving emergency telegrams . . . . .	148
4.5	CANopen and Motion Control with PROPROG wt II and motion configurator . . . . .	153
4.5.1	General information on CANopen master module and Motion Control . . . . .	153
4.5.2	Initializing CANopen master module . . . . .	153
4.5.3	Data exchange via PDOs . . . . .	153
4.5.4	Synchronizing data exchange via PDOs . . . . .	153
4.6	Structure of CANopen telegrams . . . . .	154
4.6.1	General information . . . . .	154
4.6.2	NMT telegram . . . . .	154
4.6.3	SDO telegram . . . . .	155
4.6.4	PDO telegram . . . . .	160
4.6.5	SYNC telegram . . . . .	161
4.6.6	Node Guarding telegram . . . . .	162
4.6.7	Emergency telegram . . . . .	163
	<b>Anhang A - Abbreviations . . . . .</b>	<b>165</b>
	<b>Anhang B - Tables . . . . .</b>	<b>167</b>
B.1	CANopen objects pursuant to DS 301 . . . . .	167
B.2	Default mapping of the PDO . . . . .	178
B.2.1	Default mapping for I/O modules pursuant to DS 401 . . . . .	178
B.2.2	Default mapping for drives pursuant to DSP 402 . . . . .	180
B.3	CANopen objects of the CANopen masters (software version $\geq$ 01.20) . . . . .	182
B.3.1	Overview of the objects . . . . .	182
B.3.2	Objects of the CANopen master pursuant to CiA DS 301 and DSP 302 . . . . .	184
B.3.3	Manufacturer-specific objects of the CANopen master. . . . .	208
	<b>Revision index . . . . .</b>	<b>219</b>
	<b>Index . . . . .</b>	<b>221</b>

# INTRODUCTION

This Application Manual is an important component of your b maXX system; this means that you must thoroughly read this document, not least to ensure your own safety.

In this chapter, we will describe the first steps.

## 1.1 First Steps

---

1 To program the Ethernet with CANopen-Master module for b maXX controller PLC, you need the following hardware:

- a b maXX controller PLC with power supply unit, (and if necessary other system components plugged in right hand of the power supply unit)
- an Ethernet with CANopen-Master module for b maXX controller PLC an ETH-01, ETH-02 or CAN-04 module.

You must have installed the hardware in accordance with the Operating Instructions in each case and it must be ready for operation.

2 Apart from this, you need the following software:

- ProMaster for configuration the CANopen network
- ProProg wt III for programming the b maXX controller PLC and the Ethernet with CANopen-Master module for b maXX PLC.

or

- PROPROG wt II for programming the b maXX controller PLC and the Ethernet with CANopen-Master module for b maXX PLC.

---

### NOTE



The programming system ProProg wt III is needed for programming the b maXX controller PLC BMC-M-PLC-01 / BMC-M-PLC-02 in connection with ProMaster.

---

### 1.2 Terms Used

---

We will use the term „Ethernet module“ for the „Ethernet for b maXX controller PLC“ product (BMC-M-ETH-01).

We will use the terms „Ethernet module“, „CANopen-Master“ or „CANopen-Master module“ for the „Ethernet with CANopen-Master for b maXX controller PLC“ product (BMC-M-ETH-02).

We will use the term „CANopen-Master“ for the „CANopen-Master for b maXX controller PLC“ product (BMC-M-CAN-04).

We will use the term "module" on its own when the text refers to the BMC-ETH-01, -ETH-02 or -CAN-04 products in general.

We will use the term „PLC“ for the „b maXX controller PLC“ product (BMC-PLC-01).

We will use the term „Power supply unit“ for the „Power supply unit for b maXX controller PLC“ product (BMC-PSB-01).

We will also use the term "b maXX system" for a product consisting of „b maXX controller PLC“, „Power supply unit“, „Ethernet with CANopen-Master for b maXX controller PLC“ and more components.

For a list of the abbreviations that are used, refer to [▶Appendix A - Abbreviations◀](#) from page 165 onward.

### 1.3 Conditions

---

This application manual builds on the "b maXX controller PLC Application Manual" and assumes that you have knowledge of the PROPROG wt II or ProProg wt III programming tools and have read its manual.

# BASIC SAFETY INSTRUCTIONS

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

In this chapter, we will describe the risks that can occur when working with a Baumüller 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

---



---

Hazard information will show you the dangers, that can lead to injuries or even to death. Always follow the hazard information given in this document.

---

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

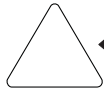
### **CAUTION**

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

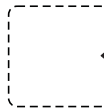
## 2.1 Hazard information and instructions

### 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.

## 2.1 Hazard information and instructions

---



### 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, over-current protection, etc.

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

## 2.4 Appropriate Use

---

You must always use the 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 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 add on the module to a b maXX controller PLC (or another system component plugged left hand of the b maXX controller PLC).
- Configure the application such that the module is always operating within its specifications.
- Ensure that only qualified personnel works with this module (or another system component plugged left hand of the b maXX controller PLC).
- Mount the module only on a b maXX controller PLC.
- Install the module as specified in this documentation.
- Ensure that connections always comply with the stipulated specifications.
- Operate the module only when it is in technically perfect condition.
- Always operate the module in an environment that is specified in the technical data.
- Always operate the module in a standard condition.  
For safety reasons, you must not make any changes to the module.
- Observe all the information on this topic if you intend to store the module.

You will be using the 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 module. We cannot, 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 added the module on another unit/module as the b maXX controller PLC (or another system component plugged left hand of the b maXX controller PLC).
- You ignored information in the operating instructions for this module.
- You did not use the module as intended.
- You handled the 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 module without written permission from Baumüller Nürnberg GmbH.
- You ignored the maintenance instructions in the component descriptions.
- You incorrectly combined the module with third-party products.
- You combined the b maXX 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 modules are protected by their packaging. Do not remove the module from its packaging until just before you intend to mount it.

The housing of the module provides IP20 protection to the modules from dirt and damage due to static discharges from contact. This means that you never use a module with damaged housing.

## 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 b maXX 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 b maXX 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 b maXX systems' place of installation, observe the applicable safety regulations for the plant in which this unit is installed.
- ▶ Provide the b maXX system with additional monitoring and protective equipment if the safety regulations demand this.
- ▶ Observe the safety measures for the unit in which the module is installed.

## 2.9 Responsibility and liability

To be able to work with this module 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 the operating instructions for this module, 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 the Operating Instructions for this module 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 Ethernet with CANopen-Master module for b maXX controller PLC 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

## ETHERNET

In this chapter, you will find information about connecting the module to an Ethernet. Ethernet is only available with the BMC-M-ETH-01 and BMC-M-ETH-02 modules.

### 3.1 General information on Ethernet and the Ethernet module

---

Ethernet is an established and proven technology for data transfer in information technology and in office communication. In the Ethernet module, a twisted pair cable is used as the transfer medium (for more information, see the module Operating Instructions). This means that you can easily implement a link to an existing 10BaseT network or to a 100BaseTX network (IEEE 802.3 standard).

You can use the Ethernet module on a network containing other components like PCs, hubs, switches and repeaters. Alternatively, you can make a direct connection to a PC's network adapter by means of a crosslink cable; in this case, you do not need any additional network components.

For communication, an Ethernet-based TCP/IP protocol stack and the

- PROPROG / OmegaOS communication application protocol are implemented.

This protocol allows you to use the following functionality for PROPROG wt II and OmegaOS:

- PLC control functions like Start and Stop
- Program management
- Online debugging, watch windows
- Logic Analyzer/oscilloscope
- OPC Server.

For further details and other functionality, refer to the Application Manual for the b maXX controller PLC and to the OPC server.

To use the functions stated above, you only need to set the communications port and the IP address in your PROPROG wt II project. On the module side, you must set the IP address, the subnet mask and the gateway or you must code them if you cannot use the default settings.

For more information on the individual steps that you have to carry out, refer to the Chapter entitled [▶Configuring Ethernet TCP/IP networks ◀](#) from page 30 onward.

The following chapters will give you an introduction to the structure of Ethernet-TCP/IP networks and the conditions for structuring communications connections between the Ethernet module and other components on the network.



---

### NOTE

The branching of Ethernet-TCP/IP networks can be very complex. There can also be a large number of components installed that are configured differently. The components influence the data traffic on the network. Apart from this, you must consider aspects of data security and unauthorized network access. This complexity means that the manual can only give you an overview in principle of using the Ethernet module on Ethernet-TCP/IP networks. To get more details about your local area network (LAN), ask your network administrator and refer to the manuals of the network components.

---

## 3.2 Basics of Ethernet Networks and TCP/IP

---

### 3.2.1 Transfer standards

---

The Ethernet module supports both of the most common physical models in the IEEE 802.3 standard:

#### 10BaseT

Each network node is connected via its own twisted-pair cable to a (star) hub that forwards all the data packets equally to all the network nodes. Eight-pin RJ45 types are used as the plug-in connectors. The maximum length of a segment (the connection between the hub and the terminal equipment) is limited to 100 meters.

#### 100BaseTX

The 100BaseTX transmission standard is similar to 10BaseT. Each network node is also connected via its own twisted-pair cable to a (star) hub that forwards all the data packets equally to all the network nodes. Eight-pin RJ45 types are also used as the plug-in connectors. Here too, the maximum length of a segment (the connection between the hub and the terminal equipment) is limited to 100 meters. However, all the components (cables, RJ45 outlets, etc.) must be rated for the higher transfer rate of 100 MHz.

The Ethernet module automatically detects the transfer rate that is being used.



### 3.2.2 Structure of Ethernet networks with a hub and switch

There are various options for structuring a star topology network.

#### All the nodes are connected via one hub.

If a node sends a data packet, it is broadcast across the entire network and is available to each of the connected nodes. The message is further-processed in each case only by the node with the correct target address. Due to the large amounts of data, collisions can occur and the system must transfer messages again.

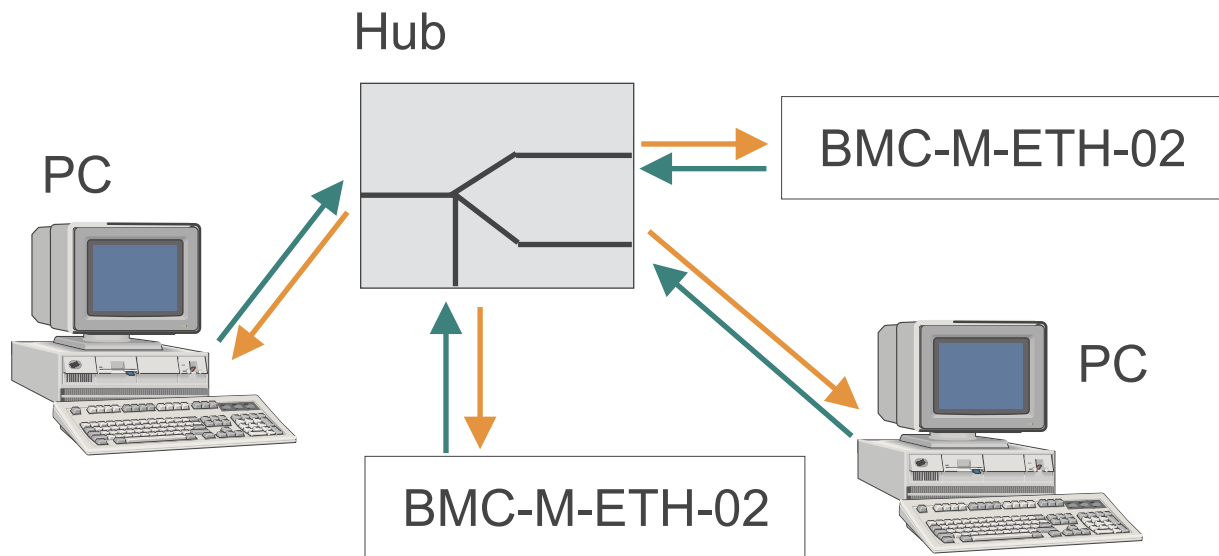


Figure 1: Principle of Shared Ethernet

#### All the nodes are connected via one switch.

In the case of a Switched Ethernet, a switch is used to connect several nodes. If data reaches the switch from a segment, the system checks the segment to which this data is to be transmitted. The system sends the data exclusively to the node with the correct target address. This reduces the amount of data and, with this, the risk of data collisions on the network.

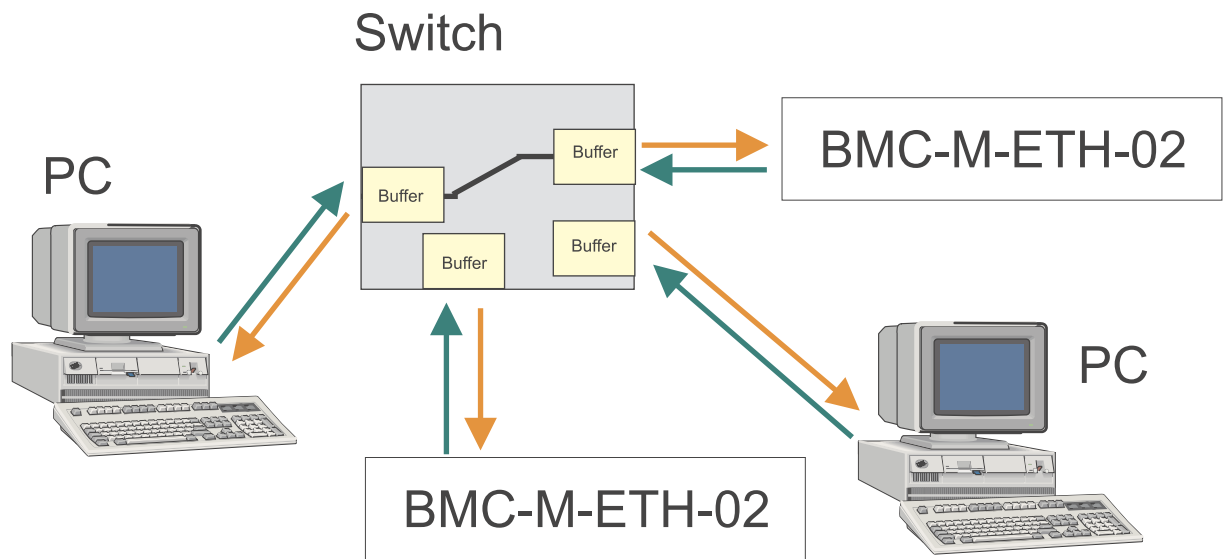


Figure 2: Principle of Switched Ethernet

By combining star structures, it is possible to form tree structures. Apart from this, it is possible to make the connection between the nodes at the TCP/IP level (see Chapter [TCP/IP](#) from page 19 onward)

### 3.2.3 Ethernet address and MAC address

Each Ethernet module has its own unique global Ethernet address. Ethernet addresses are also referred to as MAC addresses or MAC-IDs (Media Access Control Identity). The Ethernet address is permanently stored in the module and you cannot change it. Note the difference between this address and the IP address, which has a different meaning and which you can change. The MAC address is used at the Ethernet level for addressing. It has a fixed length of 6 bytes (48 bits) and contains the manufacturer code and a consecutive manufacturer number.

### 3.2.4 Ethernet data packet

Data packets are transferred on the on a connection-free basis, i.e. the sender does not get a feedback message from the receiver. The useful data is packed in a frame of address information. The structure of an Ethernet data packet of this type looks like this:

Preamble	Destination	Source	Type	Data	FCS
----------	-------------	--------	------	------	-----

Figure 3: Ethernet data packet

Preamble: A bit sequence for identifying the start of the packet  
 Destination: The receiver's Ethernet address  
 Source: The sender's Ethernet address

Type:	Indicates the higher level purpose (e.g. IP)
Data:	Useful data contains the higher level protocols, e.g. IP
FCS:	Checksum

The preamble is for synchronizing between the sending and receiving station.

The Ethernet header contains the MAC addresses of the sender and the receiver and a type field for identifying the subsequent protocol that is contained in the data area.

### 3.2.5 Bridges and Repeaters

Bridges also work at the level of the Ethernet (passing on data on the basis of the MAC-ID). Bridges connect subnets and decide on the basis of the MAC-ID the data packets that are to pass the bridge and the ones that are not to pass. By contrast with the switch, a bridge does not pass on broadcast messages (data packets to several nodes at the same time).

Repeaters are amplifiers for refreshing signals and they do not change the contents of the data packets. If a repeater detects an error on one of the connected segments, the system disconnects the connection to this segment. As soon as the error has been eliminated, the system establishes the connection again.

### 3.2.6 TCP/IP

Ethernet alone is not enough to connect several different networks for data transfer. Let us take a PC, for example, that is connected to an Internet Service Provider by means of an external modem and can use this connection to communicate with other servers on the Internet. In this case, there are other connections like the serial or USB connection to the modem and the phone line as transfer media in addition to the Ethernet. The TCP/IP protocol was developed to continue a connection outside the Ethernet too. TCP/IP is usually used as a common term, you must, however, differentiate between two different protocols.

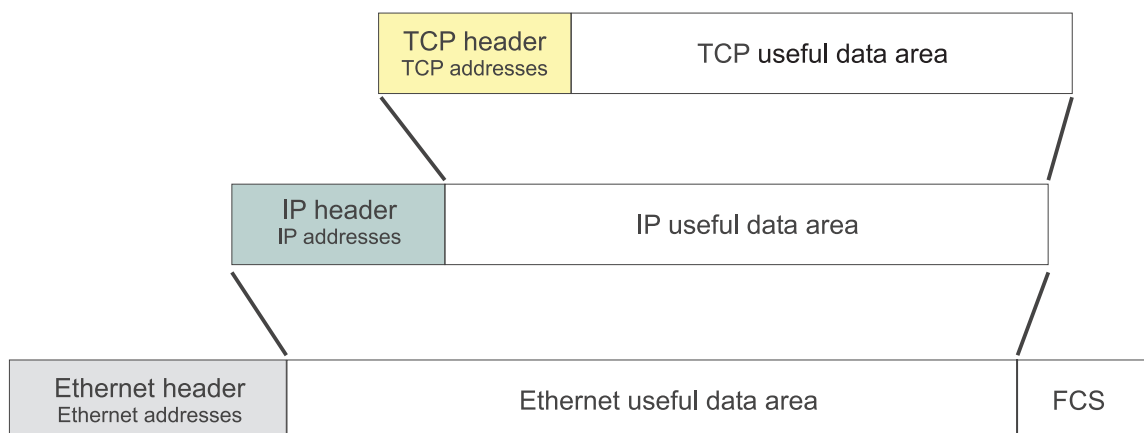


Figure 4: Structure of a TCP/IP-Ethernet data packet

### 3.2.6.1 IP protocol

---

Using **IP (Internet Protocol)**, you go beyond the limits of a LAN (local area network). IP carries out correct addressing and delivery of data packet across a gateway to other networks. IP is in the Ethernet useful data area. This means that IP packs the data that is received from higher levels into a separate frame. This packet is passed on to the Ethernet below it and it represents the useful data of one or more Ethernet message frames.

Amongst other things, this IP frame contains a new form of addresses (Internet address, IP number). Under IP, each network node has its own unique Internet address (at least in a specific subnet). This is the basis of forwarding across the limits of an Ethernet segment and linking to non-Ethernet-based networks.

#### a) Router

You connect two or more different IP networks via routers. These routers decide on the basis of the IP address whether a data packet is to be forwarded to another network or not.

#### b) IP address

You must not confuse the IP address with the MAC-ID (or Ethernet address).

Users assign the IP address. As-delivered, the Ethernet module has the IP address 192.168.1.1. For information on how to change this setting, refer to the chapter entitled [▶Configuring the Ethernet module and checking settings ◀](#) from page 39 onward.

#### c) Structure of an IP address

The IP address consists of a total of 4 bytes that are normally shown in decimal form separated by periods, e.g. 192.168.1.1.

If you use the module on a network whose nodes are only connected via a hub or switch and that does not have a connection to another network, you can assign virtually any IP address you like for each node. You must just ensure that you never set all the bits in any one byte equal to 0 or to 1 (byte = 0 or 255). These are reserved for special functions and you must not assign them. For example, you cannot use the address 194.11.0.13 due to the zero in the third byte.

If there is a connection to another network or to the Internet, you can, with the exceptions below, no longer freely assign the IP address:

For use on private networks, a corresponding IP standard reserves three address classes that may not be routed on the Internet, which means that they are not visible there. The addresses / address ranges are as follows:

- 10.xxx.xxx.xxx
- 172.16.xxx.xxx - 172.31.xxx.xxx
- 192.168.x.x

Routers on the LAN are generally set such that these address ranges are not routed. If it is not possible to communicate with an Ethernet module, this may be due to the router on your LAN not routing the set IP address. The Ethernet module's default IP address of

192.168.1.1 is an IP address of this type. Ask your network administrator which IP address range you may use if there is a connection to another network.

The address 127.0.0.1 is special, since it is always addressed as the local host. According to the standard, you are not allowed to use network address 127.x.x.x. This means that you can use address 127.0.0.1 to test the installation of your own equipment (e.g. a PC).

If you intend to connect network directly to the Internet, you can only use the unique IP addresses that are assigned by the IANA organization. Assignment depends on the country in which the network is operated.



#### NOTE

If the module is in a network that has a connection to another network or if you want to make a direct Internet connection, you will need comprehensive knowledge of the entire LAN, of assigning IP addresses, of routing mechanisms and, in particular, of security requirements. This means that you should only make a connection to another network or establish an Internet connection together with an authorized network administrator.

#### d) Subnet mask and gateway

To make a connection to another network, you must address this network. The subnet mask is used for this. If a network node is to send a data packet, its own IP address is logically ANDed with the subnet mask and the destination IP address is logically ANDed with the subnet mask. If the network node gets the same result both times, it knows that the other node is on the same network. If the results are different, it is not possible to directly address the other network node. In the case, the node passes on the data packet to a gateway or router for transmission. A gateway differs from a router inasmuch as gateways can access non-TCP/IP networks. Since the receiver of the data packet must also carry out the logical operation with the subnet mask for the response, the subnet mask must also be set correctly on the receiver side.



#### NOTE

Since a router is only a special gateway, when configuring the IP address people often use the term "gateway" even though a router is physically present. This is particularly the case with the network configuration in Windows. With the Ethernet module, too, we use the more general term "Gateway".

Users assign the subnet mask and the gateway or router IP address. For information on how to make these settings in the Ethernet module, refer to the chapter entitled [▶Configuring the Ethernet module and checking settings](#) ◀ from page 39 onward. The default settings are as follows:

- Subnet mask: 255.255.255.0
- Gateway: 0.0.0.0

The gateway IP address 0.0.0.0 with the Ethernet module means that no gateway is used. This means that the data packet is also sent even if the system detects that due to the subnet mask link the receiver is not on the same network.

### e) Subnet mask and network class

The InterNIC (International Network Information Center) divided the IP address into a "network section" and a "host section" to create what are known as address classes. The table below shows the different address classes, the assigned values of the most significant bit of the IP address and the division into the "network section" and the "host section".

Address class	Description	Address range of the network section	Possible number of hosts
Class A	The first byte of the IP address is for addressing the network section; the last three bytes address the host section	1.xxx.xxx.xxx to 126.xxx.xxx.xxx	Approximately 16 million
Class B	The first two bytes of the IP address are for addressing the network section; the last two bytes address the host section	128.0.xxx.xxx - 191.255.xxx.xxx	Approximately 65 thousand
Class C	The first three bytes of the IP address are for addressing the network section; the last byte addresses the host section	192.0.0.xxx to 223.255.255.xxx	254

In addition, there are Class D and Class E networks that are of no great significance in practical applications.

This subdivision also affects the subnet mask. Depending on the address class to which an IP address belongs, the subnet mask has a minimum value that is dependent on the permitted range of the IP address's "network section".

Address class	Minimum subnet mask
Class A	255.0.0.0
Class B	255.255.0.0
Class C	255.255.255.0

The Ethernet module complies with these address class-dependent subnet mask, i.e. if the subnet mask that a user enters is less than the address class mask that belongs to the IP address, the system uses the address class mask.

#### Example:

A PC with PROPROP wt II has an IP address of 192.075.191.188; an Ethernet module has an IP address of 192.168.1.1. Both network nodes are located physically on the same network.

Since only the first bytes of both IP addresses are identical, you should choose 255.0.0.0 as the subnet mask on both systems to get the same result of logically ANDing the "IP address AND the subnet mask". However, since the IP addresses belong to a Class C network, the module uses subnet mask 255.255.255.0. Logically ANDing the "IP address AND the subnet mask" yields different networks: 192.075.191.0 and 192.168.1.0. This

means that communication is not possible, since the data packets are transferred to the set gateway.

You can avoid this problem and establish communication by deactivating the gateway and assigning gateway IP address 0.0.0.0.

#### f) Examples of IP networks

Example 1:

Network A  
192.168.1.x

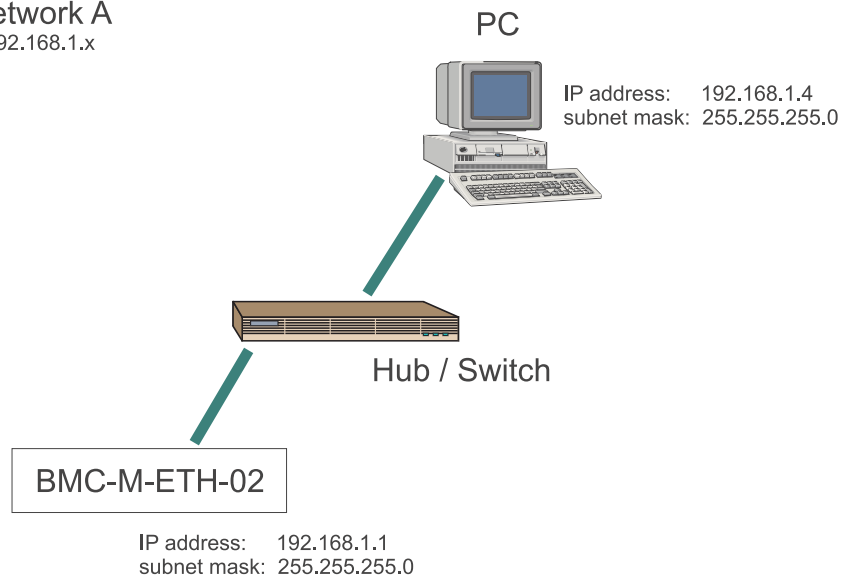


Figure 5: Example: Module and PC on the same IP network

On network A, the components have the stated settings. Communication is possible between the PC and the module, since logically ANDing both components "IP address AND subnet mask" yields the same network: 192.168.1.0

Example 2:

Network A

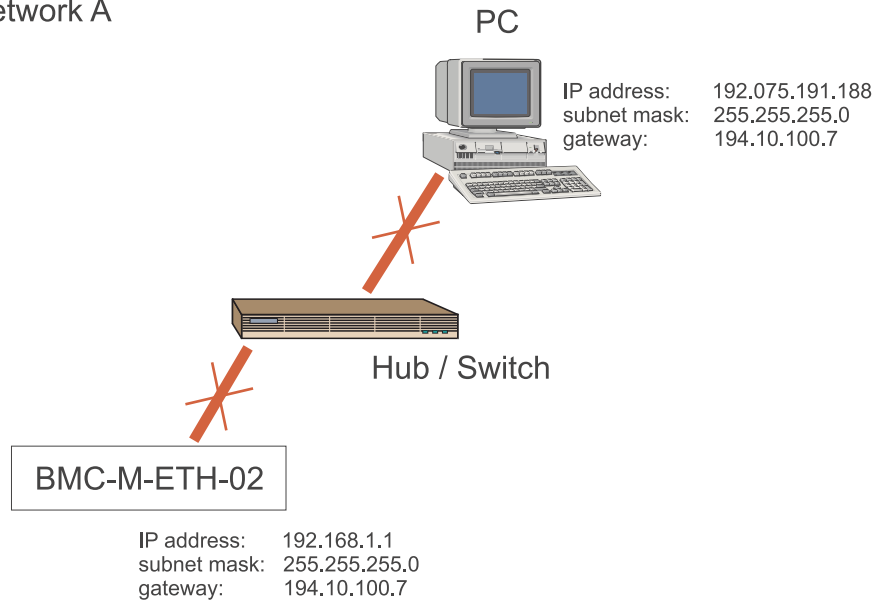


Figure 6: Example: Module and PC on different IP networks

On network A, the components have the stated settings. Communication between the PC and the module is not possible, since logically ANDing both components "IP address AND subnet mask" yields different networks. The PC and the module would transfer the data packet to a gateway.

Example 3:

Network A

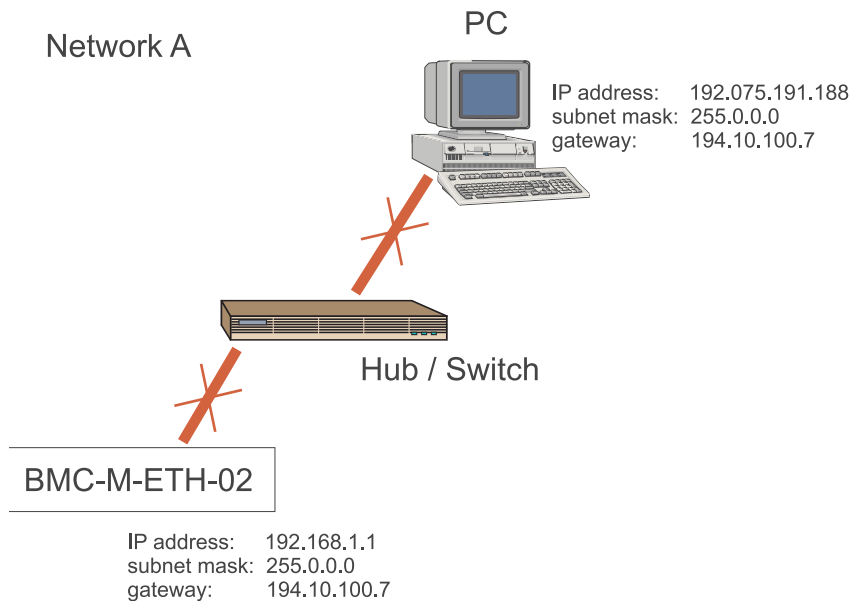


Figure 7: Example: Module and PC with the wrong subnet mask



On network A, the components have the stated settings. Communication between the PC and the module is not possible. Logically ANDing the "IP address AND the subnet mask" yields the same network 192.x.x.x. However, the module detects a Class C network and uses subnet mask 255.255.255.0. and would transfer the data packets to a gateway. For a remedy, see example 4.

Example 4:

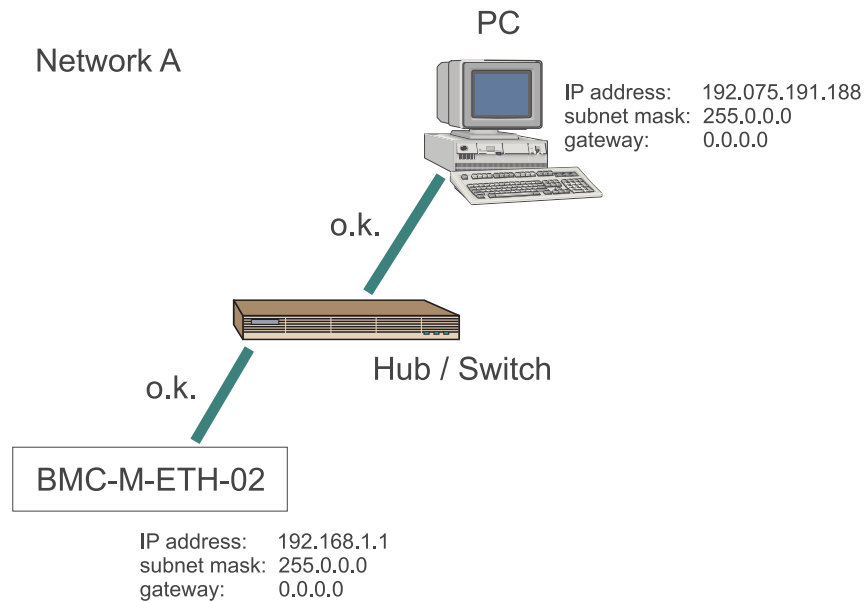


Figure 8: Example: Module and PC without a gateway

On network A, the components have the stated settings. Communication between the PC and the module is possible. Logically ANDing the "IP address AND the subnet mask" yields the network 192.x.x.x. However, the module detects a Class C network and uses subnet mask 255.255.255.0. and would transfer the data packets to a gateway. However, since no gateway is set, the system does not transfer any data.

Example 5:

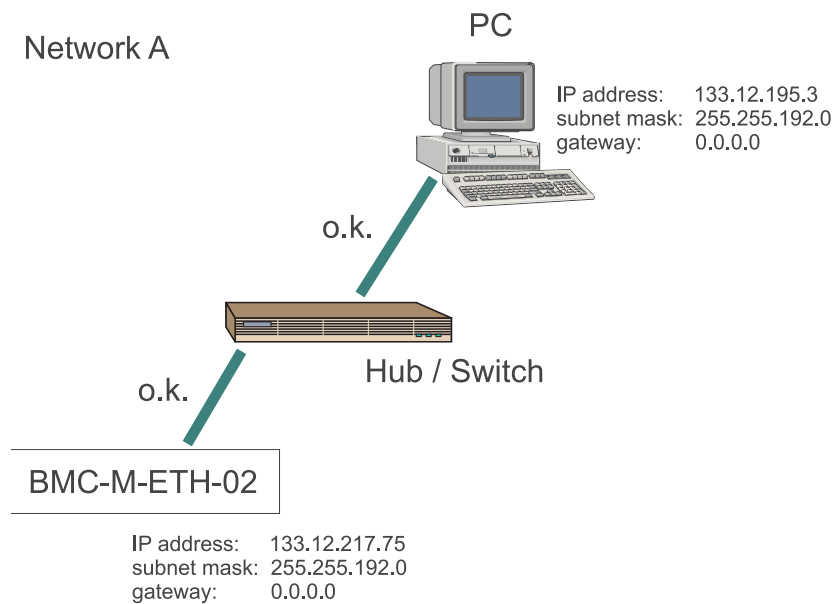


Figure 9: Example: Module and PC on a Class-B IP network

On network A, the components have the stated settings. Communication is possible between the PC and the module, since logically ANDing both components "IP address AND subnet mask" yields the same network: 133.12.192.0

Example 6:

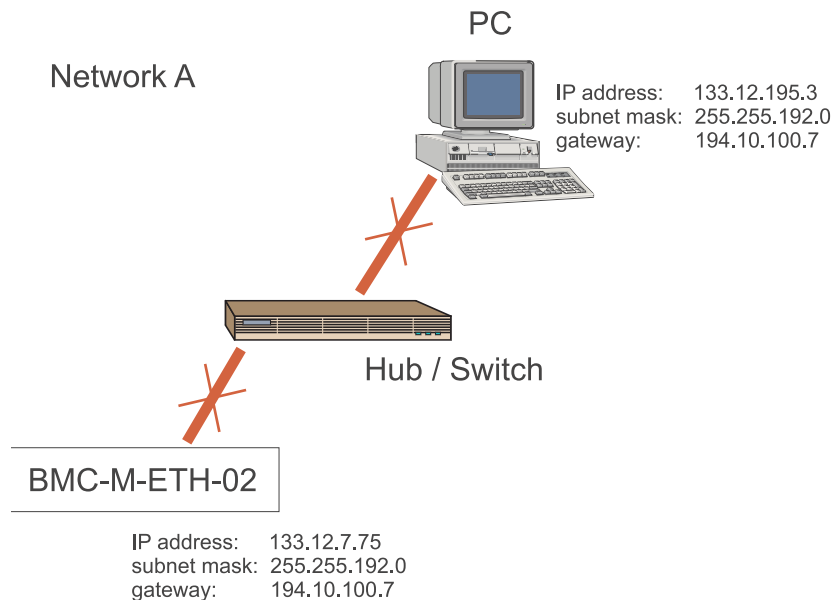


Figure 10: Example: Module and PC on different Class-B IP networks

On network A, the components have the stated settings. Communication between the PC and the module is not possible, since logically ANDing "the IP address AND the subnet

mask" yields different networks: 133.12.192.0 and 133.12.0.0. The system would transfer the data packets to a gateway.

Example 7:

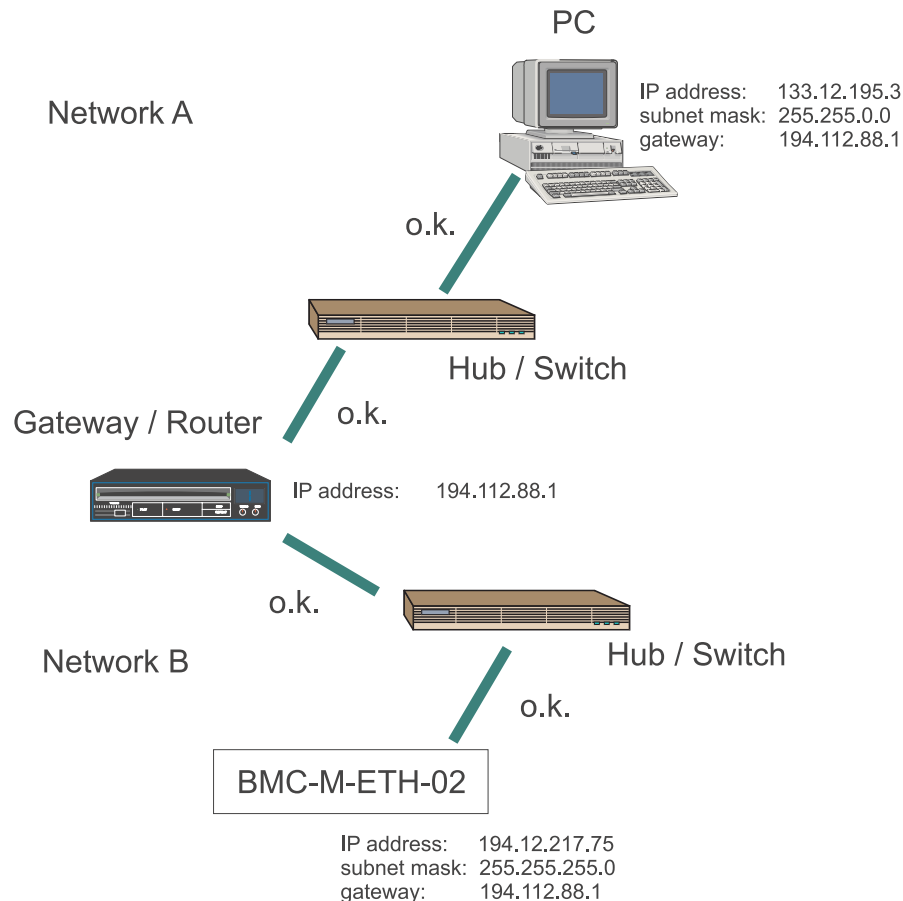


Figure 11: Example: Module and PC with a gateway

On network A and network B, the components have the stated settings. The networks are connected via a gateway with IP address 194.112.88.1. Communication between the PC and the module is possible, since logically ANDing both components with "IP address AND subnet mask" yields different networks – 133.12.0.0 and 194.12.217.0 – but both transfer data packets to the same gateway that carries out further- distribution.

### 3.2.6.2 TCP protocol

The **TCP (Transport Control Protocol)** is responsible for transporting and saving data and is located in the IP useful data area. TCP works on the client server principle. For the duration of the data transfer, it makes a connection between two network nodes that is known as a session. Within any one session, the system divides up useful data to individual numbered TCP packets and puts them back together again such that if necessary they can be transferred over different paths with different runtimes. Checksums make it

possible to check and confirm correct reception. The system detects that individual TCP packets have been lost and it can request them again.

TCP packets are inserted in the useful data area of an IP packet. Apart from this, TCP forwards the useful data to the correct application program on the destination computer by assigning different port numbers to different applications (services).

The port number for die PROPROG communication is 0x5043 (20547 dec.). The originating port is the port number of the sending application and at the same time the return sending port of the response.

As a user, you do not need to make any settings at the TCP level.

### 3.2.7 Interaction between Ethernet and TCP/IP, ARP

---

Up to now, addressing the nodes is only ensured by the IP addresses. At the Ethernet level, the network node only knows its own MAC-ID. To be able to deliver the IP packet in an Ethernet frame, however, the node must know the receiver's MAC-ID. The ARP protocol (Address Resolution Protocol) was developed to determine this. To find out the pair of Ethernet IP addresses of a network node, the ARP protocol sends an Ethernet broadcast containing the known IP address of the receiver. The receiver that has this IP address replies to the requesting system with a packet that contains the pair of Ethernet IP addresses. To avoid unnecessary ARP requests, the reply packet is stored on the requesting system in an ARP table. To keep this ARP table as small as possible, the system deletes the entries at specific, system-dependent time intervals. From the command line, you can read out the current ARP table for your PC:

```
arp -a
```

If communication has already taken place with the Ethernet module, you will find the module in the listing. To trigger communication, you can use the ping command from the command line.

```
ping 192.168.1.1
```

Assuming that your network is correctly configured (the IP addresses, subnet masks and the gateways), the module with IP address 192.168.1.1 will reply in the following form:

```
Reply from 192.168.1.1: Bytes.....
```

If you do not get a reply, check your network settings.

If you do get a reply, the module is now also listed in your PC's ARP table. It is not possible to read out the Ethernet module's ARP table.

#### **If you change the module note the following:**

The PC's ARP table contains the mapping IP address ↔ MAC address of the module. The ARP table is not updated continuously.

If a module is changed, the MAC address changes too (which is linked to the hardware). For this reason the PC's ARP table doesn't agree anymore with the mapping IP address ↔ MAC address of the module immediately after the change. There is no communication between PC and module.

In this case you must either maintain the ARP table until it is updated or the PC's ARP table is deleted. In order to delete the ARP table use the following command line by entering:

```
arp -d *
```

The ARP table is generated new after it has been deleted, IP address ↔ MAC address assignment of the (changed) module are suitable and the communication between PC and module is possible.

### 3.2.8 Proxy

A proxy is a computer that accepts requests from a network node and forwards them to the intended destination. The destination transfers the result of the request to the proxy, which, for its part, can return the result to the other network node. A proxy can work on different application levels and check the data to be transferred. It often has a large amount of memory (a proxy cache) to buffer requested data and, in the case of a possible subsequent request from a different network node or from the same one, to make available the data to the actual destination on a direct basis, i.e. without needing an additional request. A firewall proxy is generally used if an internal network, e.g. an Intranet/LAN is intended to make a protected connection to another network (e.g. the Internet). In this connection, the proxy acts as a kind of gateway with IP and packet filters. On the basis of the origin, destination, port and packet type information that each packet contains, the proxy can inspect data transfer from one network to another one and if necessary it can limit it. Using appropriate software functions, the proxy can carry out further security tasks above the IP level.

### 3.2.9 Application layer PROPROG wt II / OmegaOS

PROPROG wt II / OmegaOS data packets are transferred in the useful data area of TCP packets. Users do not need to make any further settings here. You must only set the communications port in PROPROG wt II (see the chapter entitled [▶Setting a connection via Ethernet-TCP/IP in PROPROG wt II / ProProg wt III](#) ◀ from page 47 onward). The Ethernet module makes available two communications channels; i.e. using the module you can operate in-parallel a maximum of

two ProMaster

or

ProMaster and ProProg wt III

or

two ProProg wt III

or

ProMaster (or ProProg wt III) and one OPC application

or

two OPC applications

OR

two PROPROG wt II

or

one PROPROG wt II and one OPC application

or

two OPC applications

### 3.3 Configuring Ethernet TCP/IP networks

---

#### 3.3.1 Overview

---

To be able to use the Ethernet module via Ethernet with TCP/IP using PROPROG wt II or via OPC, you must carry out the following steps:

- physically commission the network (see the Operating Instructions of module BMC-M-ETH-01 / BMC-M-ETH-02 for b maXX controller PLC and the Operating Instructions of the network components)
- specify the network data: the IP addresses and subnet masks (see the previous chapter [▶TCP/IP ◀](#) from page 19 onward)
- set IP addresses, subnet masks and gateways in the network components (see the next chapter)
- set the communications port in PROPROG wt II (see the next chapter entitled [▶Setting a connection via Ethernet-TCP/IP in PROPROG wt II / ProProg wt III ◀](#) from page 47 onward)

#### 3.3.2 Configuring the Windows PC

---

Since PROPROG wt II and the OPC server need a Windows operating system, we will only describe the settings that are necessary for Windows operating systems. The basic condition is that the TCP/IP protocol is installed on your Windows PC and that your computer has a configured network adapter. For details about this installation, refer to the Windows documentation.

##### 3.3.2.1 Configuring TCP/IP under Windows XP

---

- Click on the *Start* pushbutton, and then choose My Network Places. Click on *Network Tasks* to View *Network Connections*.
- Double-click on the *Local Area Connection* icon and click on the *Properties* → pushbutton; the system displays a list of the components that use your network adapter.

- Check *Internet Protocol (TCP/IP)* and click on the *Properties* pushbutton.

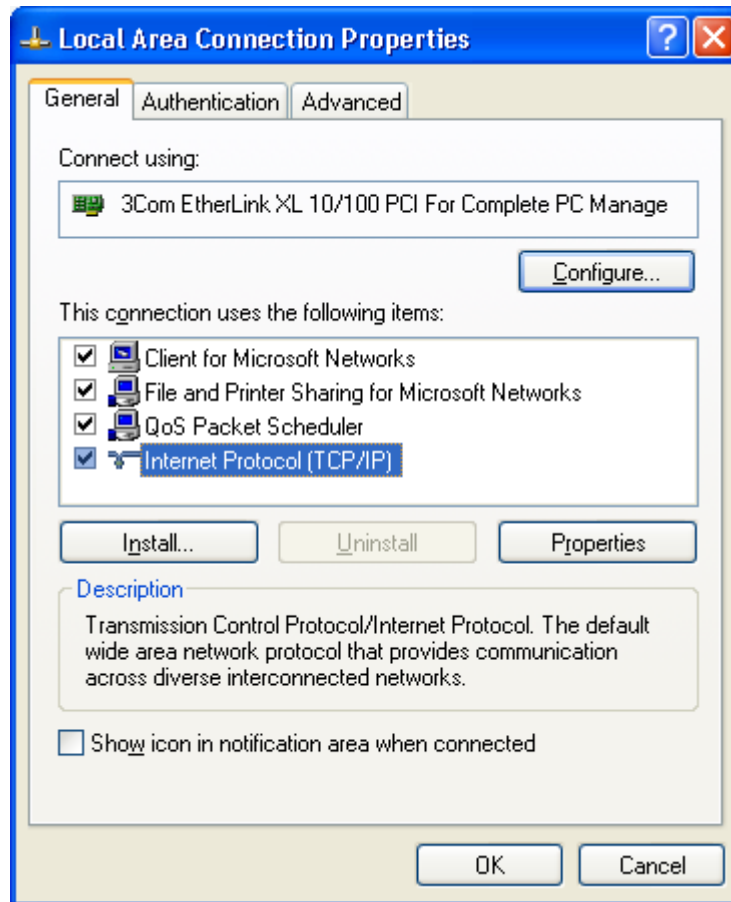


Figure 12: Setting up Windows XP with TCP/IP – overview of LAN connections

- You can now choose between *Obtain an IP address automatically* or *Use the following IP address*. If you obtain the IP address automatically, you can query the current setting either in field *Details* of *LAN connection* in the *Network Connections* window, or

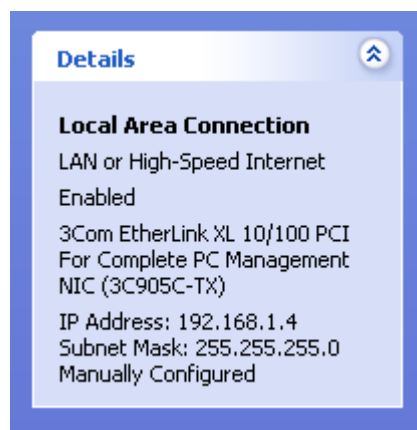


Figure 13: Setting up Windows XP with TCP/IP – details of LAN connections

on the command line by entering

```
ipconfig
```

If the settings match the ones you specified before, you can end the dialog and configuration. Note, however, that the system makes these settings dynamically, i.e. the next time you start your PC, this may result in different settings.

Check *Use the following IP address* if you want to enter special values for the IP address, the subnet mask and the gateway:

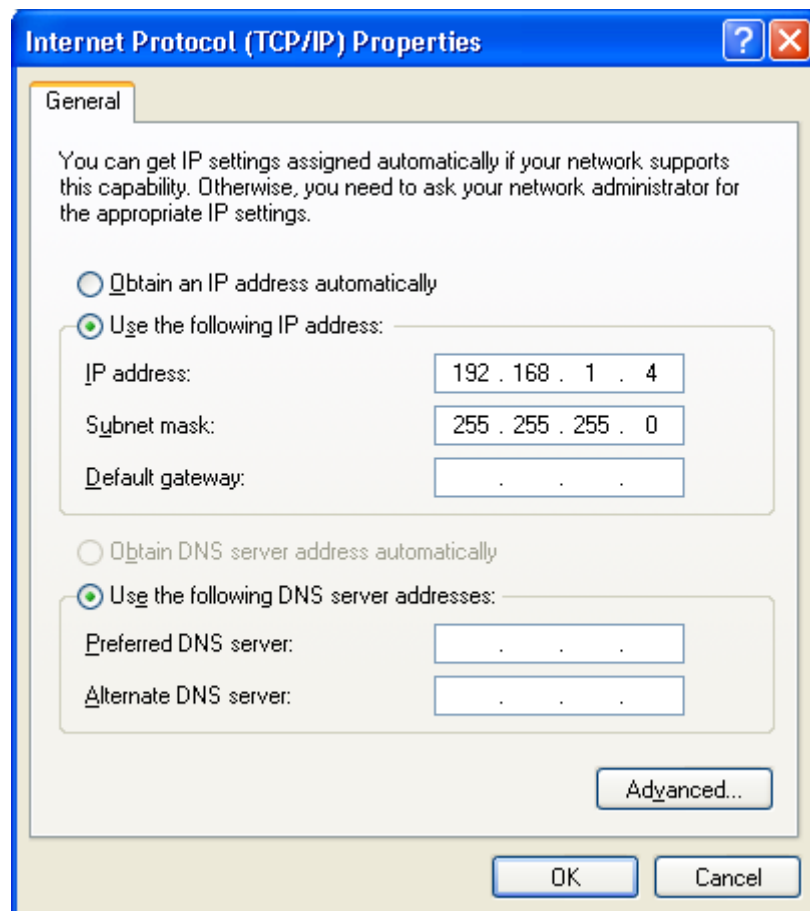


Figure 14: Setting up Windows XP with TCP/IP – TCP/IP properties

- Close the windows that you opened for configuration by clicking on the *OK* pushbutton. This completes installation of TCP/IP support on your Windows XP system. Under Windows XP, you do not need to reboot your computer.



### 3.3.2.2 Configuring TCP/IP under Windows 2000

- Click on the *Start* pushbutton and under *Settings* open *Network and Dial-up Connections*
- Double-click on the *Local Area Connection* icon and click on the *Properties* → pushbutton; the system displays a list of the components that use your network adapter.
- Check *Internet Protocol (TCP/IP)* and click on the *Properties* pushbutton.

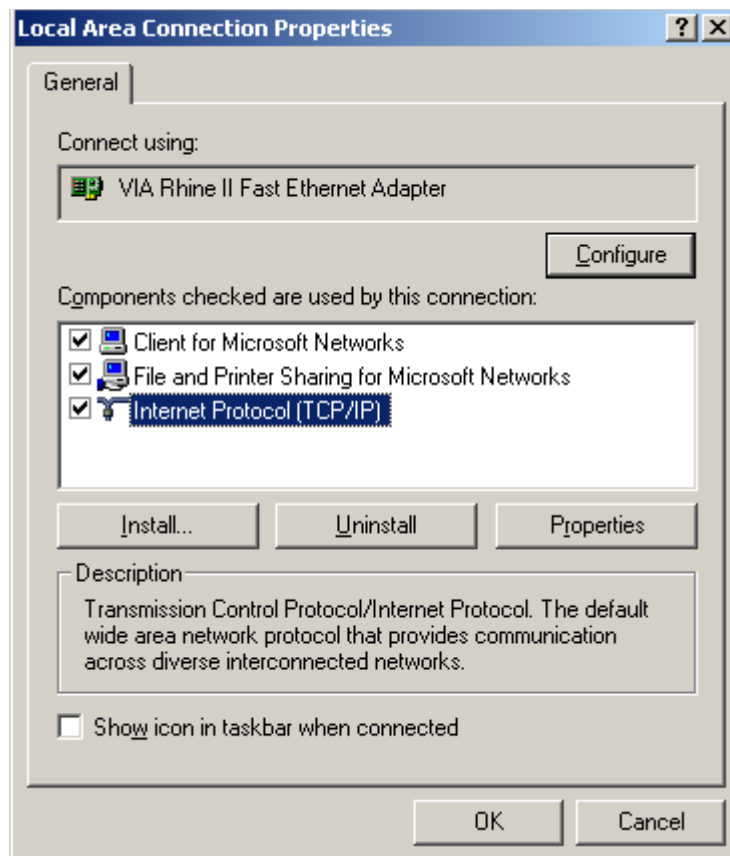


Figure 15: Setting up Windows 2000 with TCP/IP – overview of LAN connections

- You can now choose between *Obtain IP address automatically* or *Use the following IP address*. If you want to *Obtain IP address automatically*, you can now query the current setting on the command line by entering

```
ipconfig
```

If the settings match the ones you specified before, you can end the dialog and configuration. Note, however, that the system makes these settings dynamically, i.e. the next time you start your PC, this may result in different settings.

Check *Use the following IP address* if you want to enter special values for the IP address, the subnet mask and the gateway:

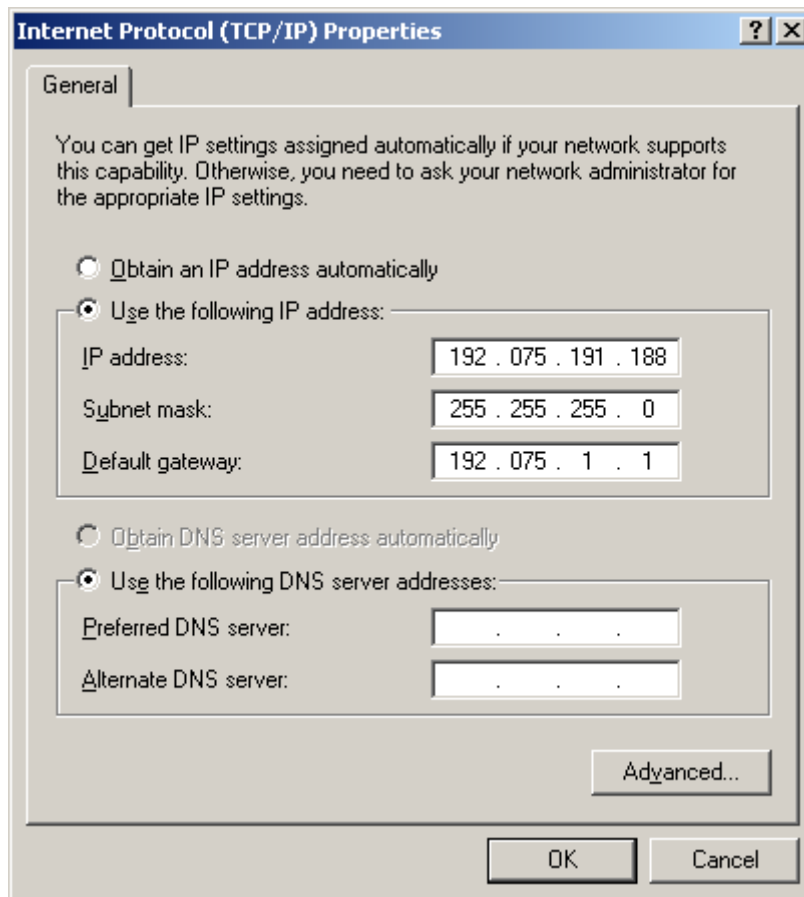


Figure 16: Setting up Windows 2000 with TCP/IP – TCP/IP properties

- Close the windows that you opened for configuration by clicking on the *OK* pushbutton. This completes installation of TCP/IP support on your Windows 2000 system. Under Windows 2000, you do not need to reboot your computer.

### 3.3.2.3 Configuring TCP/IP under Windows NT 4.0

- Click on the *Start* pushbutton and under *Settings* open the *Control Panel*
- Double-click on the *Network* icon and choose the *Protocols* tab  
→ The system displays the installed network protocols
- Check *TCP/IP protocol* and click on the *Properties* pushbutton.

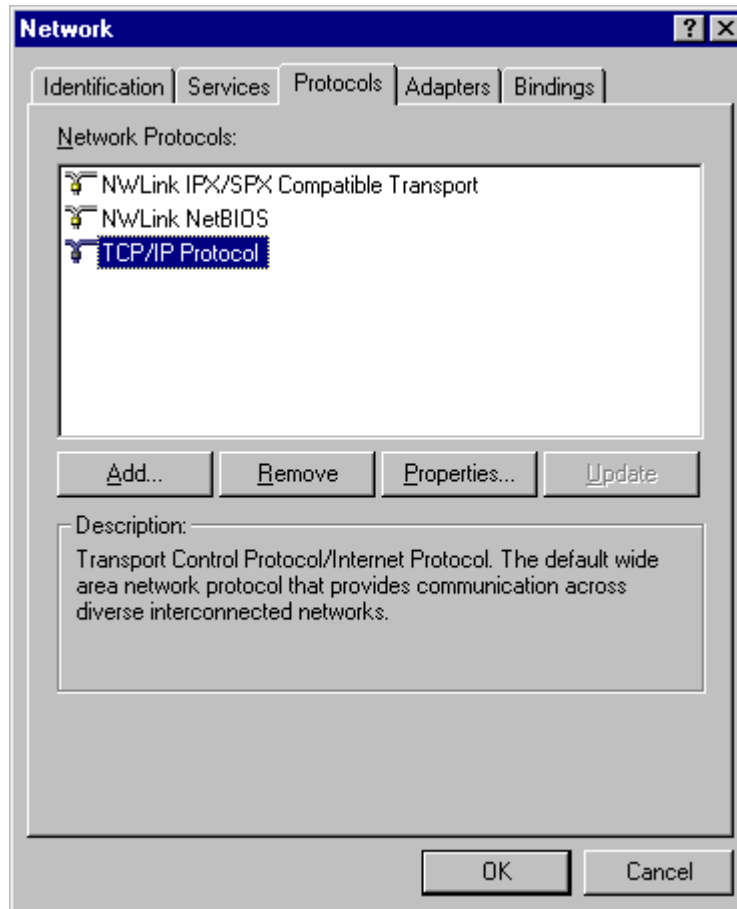


Figure 17: Setting up Windows NT 4.0 with TCP/IP – overview of LAN protocols

- You can now choose between *Obtain an IP address from a DHCP server* or *Specify an IP address*. If you choose to obtain an IP address from a DHCP server, you can query the current setting from the command line by entering

```
ipconfig
```

If the settings match the ones you specified before, you can end the dialog and configuration. Note, however, that the system makes these settings dynamically, i.e. the next time you start your PC, this may result in different settings.

Check *Specify an IP address* if you want to enter special values for the IP address, the subnet mask and the gateway:

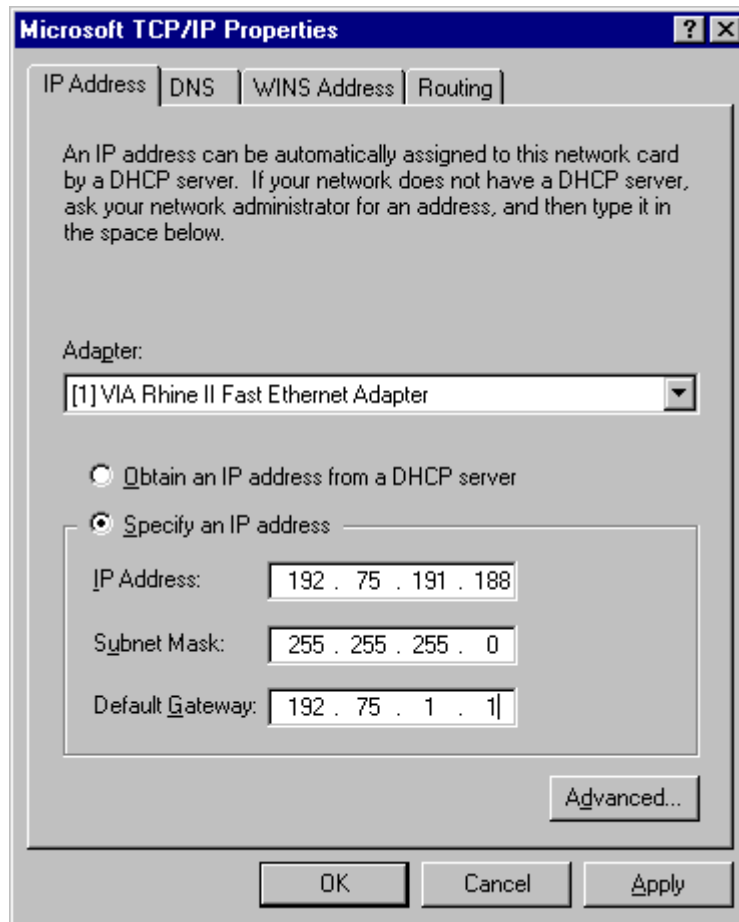


Figure 18: Setting up Windows NT 4.0 with TCP/IP – TCP/IP properties

- Close the windows that you opened for configuration by clicking on the *OK* pushbutton. This completes installation of TCP/IP support on your Windows NT 4.0 system. Under Windows NT 4.0, you must now reboot your computer!

### 3.3.2.4 Configuring TCP/IP under Windows 9x

- Click on the *Start* pushbutton and under *Settings* open the *Control Panel*
- Double-click on the *Network* icon and choose the *Configuration* tab  
→ The system displays the installed network components
- Check *TCP/IP* and click on the *Properties* pushbutton.

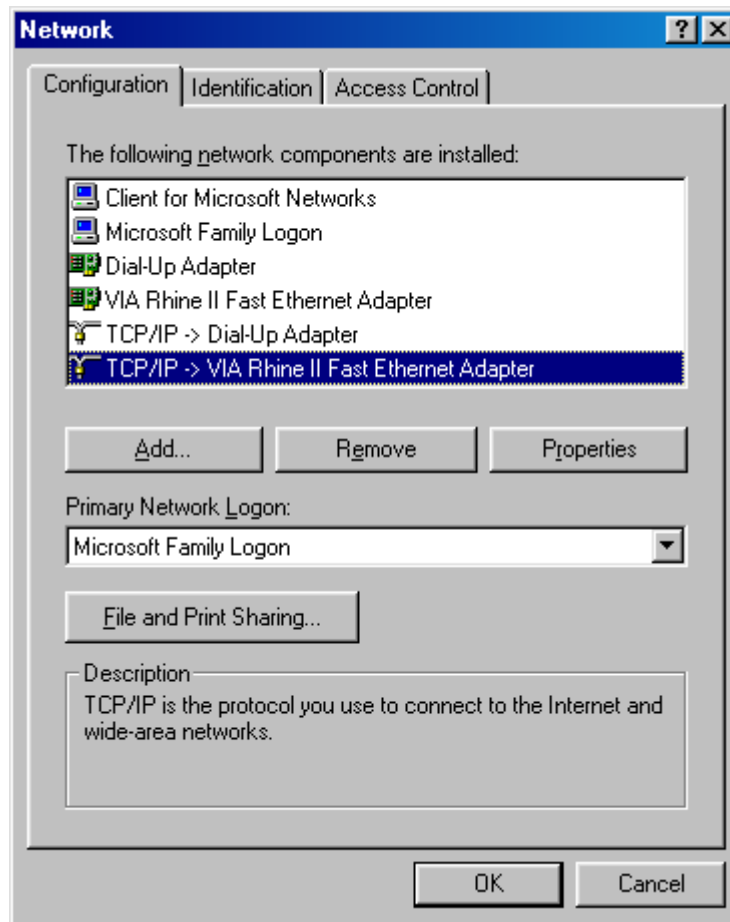


Figure 19: Setting up Windows 9x with TCP/IP – overview of LAN configuration

- Choose the *IP address* tab
- You can now choose between *Obtain an IP address automatically* or *Specify an IP address*. If you want to *Obtain an IP address automatically*, you can now query the current setting on the command line by entering

```
winipcfg
```

If the settings match the ones you specified before, you can end the dialog for assigning the IP address. Note, however, that the system makes these settings dynamically, i.e. the next time you start your PC, this may result in different settings.

Check *Specify an IP address* if you want to enter special values for the IP address, the subnet mask and the gateway:

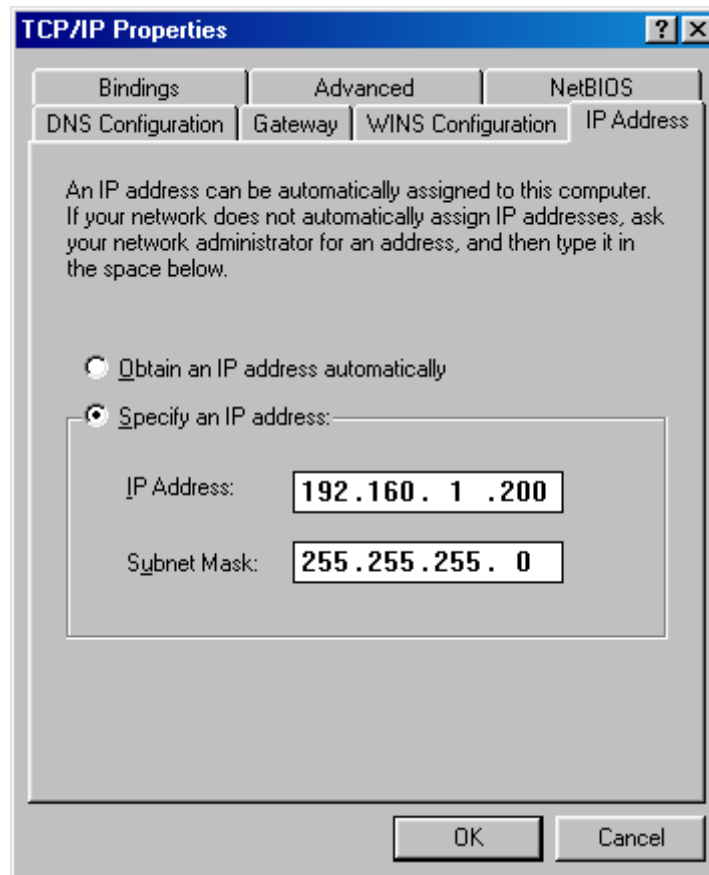


Figure 20: Setting up Windows 9x with TCP/IP – IP address

- Choose the *Gateway* tab  
In field *New gateway*, enter the IP address of the gateway if you want to use a special value for it:

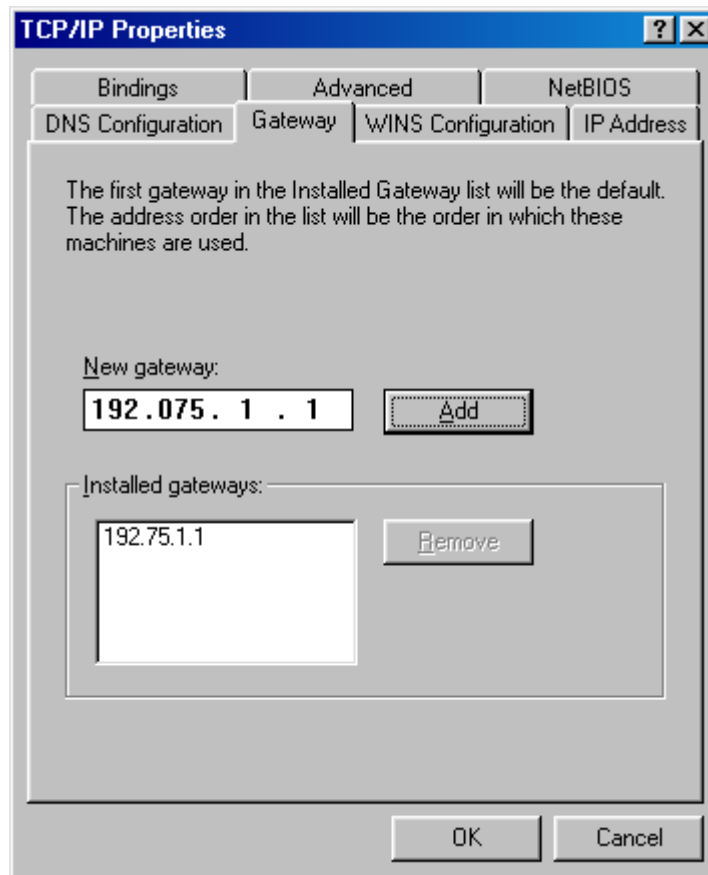


Figure 21: Setting up Windows 9x with TCP/IP – gateway

- Click on the *Add* pushbutton. The system takes the gateway's address in the list of installed gateways.
- Close the windows that you opened for configuration by clicking on the *OK* pushbutton.

This completes installation of TCP/IP support on your Windows 9x system. Under Windows 9x, you must reboot your computer.

### 3.3.3 Configuring the Ethernet module and checking settings

You have two options for configuring the Ethernet module for TCP/IP communication:

- by using the default settings
- by freely configuring using PROPROG wt II / ProProg wt III

#### 3.3.3.1 Default settings for TCP/IP

The default settings for TCP/IP communication of the Ethernet option module are:

IP address:	192.168.1.1 + "Rotary switch"
Subnet mask:	255.255.255.0
Gateway:	0.0.0.0

## 3.3 Configuring Ethernet TCP/IP networks

Using rotary switches S2 and S3 (on the module), you can set IP addresses in the range 192.168.1.1 to 192.168.1.32.

In order to set the rotary switches of the module see [▶ Operating Instructions Ethernet with CANopen-Master for b maXX controller PLC](#).

### 3.3.3.2 Freely configuring TCP/IP

a) Prepare for configuration using the PROPROGRAM wt II / ProProg wt III

You can also freely set the values for the IP address, the subnet mask and the gateway. To do this, proceed as follows:

- 1 Create a PROPROGRAM wt II / ProProg wt III project for the b maXX controller PLC if you have not already created a separate project for your application.
- 2 PROPROGRAM wt II: In this project, link library BM\_TYPES\_20bd06 (or higher) if it is not already present.  
ProProg wt III: In this project, link library BM\_TYPES\_30bd01 (or higher) if it is not already present.
- 3 Create a POU that can later be called in a cold boot and warm restart task, assuming that one is not present.
- 4 Create a global variable of data type ETHERNET\_PLC\_CONFIG\_BMSTRUCT. You must connect this global variable to the base address to the Ethernet configuration. The base address depends on the module number (01 to 05, set on S1). The following module numbers are possible:

Module number (S1)	Base address for Ethernet configuration
01	%MB3.2012288
02	%MB3.3012288
03	%MB3.4012288
04	%MB3.5012288
05	%MB3.6012288
06 to 15	reserved

Example if the Ethernet module has the module number 01 (S1 = 1):

PROPROGRAM wt II:

```
VAR_GLOBAL
_EthernetConfig_MNr_01 AT %MB3.2012288 : ETHERNET_PLC_CONFIG_BMSTRUCT;
END_VAR
```

Figure 22: Example: Global variable of type ETHERNET\_PLC\_CONFIG\_BMSTRUCT (PROPROGRAM wt II)



ProProg wt III:

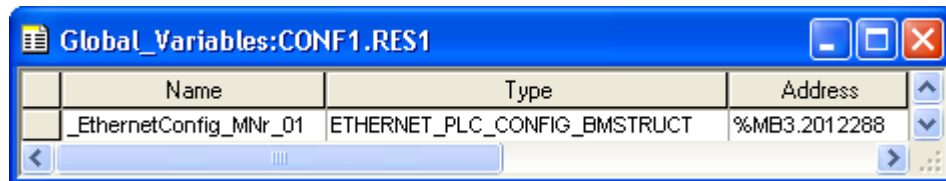


Figure 23: Example: Global variable of type ETHERNET\_PLC\_CONFIG\_BMSTRUCT (ProProg wt III)

Where:

`_EthernetConfig_MNr_01` is the variable name with data type short designation "\_" for STRUCT

`%MB3.2012288` is the base address for Ethernet configuration of the module with module number 01

`ETHERNET_PLC_CONFIG_BMSTRUCT` is the data type of the variable

- 5** To set the IP address, subnet mask and gateway, you need structure elements `a_IP_ADDRESS`, `a_IP_MASK` and `a_GATEWAY` of the created global variable. All three elements consist of one field (array) containing four entries each of data type `USINT`. Each address starts in field index 0 with the network section. To activate the configuration, you additionally need structure element `d_IP_CONFIG` of data type `DWORD`. Create these elements in the POU for cold boot/warm restart.

Example:

(\* Configuration of IP-Address \*)

```
◆_EthernetConfig_MNr_01.a_IP_ADDRESS[0]◆  
◆_EthernetConfig_MNr_01.a_IP_ADDRESS[1]◆  
◆_EthernetConfig_MNr_01.a_IP_ADDRESS[2]◆  
◆_EthernetConfig_MNr_01.a_IP_ADDRESS[3]◆
```

(\* Configuration of Subnetmask \*)

```
◆_EthernetConfig_MNr_01.a_IP_MASK[0]◆  
◆_EthernetConfig_MNr_01.a_IP_MASK[1]◆  
◆_EthernetConfig_MNr_01.a_IP_MASK[2]◆  
◆_EthernetConfig_MNr_01.a_IP_MASK[3]◆
```

(\* Configuration of Gateway \*)

```
◆_EthernetConfig_MNr_01.a_GATEWAY[0]◆  
◆_EthernetConfig_MNr_01.a_GATEWAY[1]◆  
◆_EthernetConfig_MNr_01.a_GATEWAY[2]◆  
◆_EthernetConfig_MNr_01.a_GATEWAY[3]◆
```

(\* The configuration command \*)

```
◆_EthernetConfig_MNr_01.d_IP_CONFIG◆
```

Figure 24: Example: Elements of a global variable of type ETHERNET\_PLC\_CONFIG\_BMSTRUCT

### 6 There are two options for configuring the IP address:

A fixed IP address or a rotary switch-dependent IP address. In the first case, you set an IP address that is independent of the rotary switches S2, S3. In the second case, you add the value of the rotary switches S2 and S3 to the configured IP address. The value of element d\_IP\_CONFIG decides which procedure is to be used.

#### b) Assigning a fixed IP address

To assign a fixed IP address that is independent of the rotary switches, you assign the structure elements of the global variable for Ethernet configuration with the required values. Finally, you must write element d\_IP\_CONFIG with DWORD#16#12345678. When writing to the elements, you must absolutely comply with the following sequence

a\_IP\_ADDRESS → a\_IP\_MASK → a\_GATEWAY → d\_IP\_CONFIG

Example:

```
(* Configuration of IP-Address*)
USINT #133————_EthernetConfig_MNr_01.a_IP_ADDRESS[0]
USINT #12————_EthernetConfig_MNr_01.a_IP_ADDRESS[1]
USINT #195————_EthernetConfig_MNr_01.a_IP_ADDRESS[2]
USINT #3————_EthernetConfig_MNr_01.a_IP_ADDRESS[3]

(* Configuration of Subnetmask *)
USINT #255————_EthernetConfig_MNr_01.a_IP_MASK[0]
USINT #255————_EthernetConfig_MNr_01.a_IP_MASK[1]
USINT #192————_EthernetConfig_MNr_01.a_IP_MASK[2]
USINT #0————_EthernetConfig_MNr_01.a_IP_MASK[3]

(* Configuration of Gateway *)
USINT #0————_EthernetConfig_MNr_01.a_GATEWAY[0]
USINT #0————_EthernetConfig_MNr_01.a_GATEWAY[1]
USINT #0————_EthernetConfig_MNr_01.a_GATEWAY[2]
USINT #0————_EthernetConfig_MNr_01.a_GATEWAY[3]

(* The configuration command *)

DWORD#16#12345678—_EthernetConfig_MNr_01.d_IP_CONFIG
```

Figure 25: Example: Assigning a fixed IP address, gateway and subnet mask for the module

For the Ethernet module, this yields the IP address 133.12.195.3 and the subnet mask 255.255.192.0. The gateway has the address 0.0.0.0, i.e. no gateway is used.

#### c) Assigning a variable, rotary switch-dependent IP address

To assign a variable IP address that is dependent on the rotary switches, you assign the structure elements of the global variable for Ethernet configuration with the required values. Finally, you must write element `d_IP_CONFIG` with `DWORD#16#12345600`. When writing to the elements, you must absolutely comply with the following sequence

`a_IP_ADDRESS` → `a_IP_MASK` → `a_GATEWAY` → `d_IP_CONFIG`

Example:

```
(* Configuration of IP-Address*)
USINT #133—————_EthernetConfig_MNr_01.a_IP_ADDRESS[0]
USINT #12—————_EthernetConfig_MNr_01.a_IP_ADDRESS[1]
USINT #195—————_EthernetConfig_MNr_01.a_IP_ADDRESS[2]
USINT #3—————_EthernetConfig_MNr_01.a_IP_ADDRESS[3]

(* Configuration of Subnetmask *)
USINT #255—————_EthernetConfig_MNr_01.a_IP_MASK[0]
USINT #255—————_EthernetConfig_MNr_01.a_IP_MASK[1]
USINT #192—————_EthernetConfig_MNr_01.a_IP_MASK[2]
USINT #0—————_EthernetConfig_MNr_01.a_IP_MASK[3]

(* Configuration of Gateway *)
USINT #0—————_EthernetConfig_MNr_01.a_GATEWAY[0]
USINT #0—————_EthernetConfig_MNr_01.a_GATEWAY[1]
USINT #0—————_EthernetConfig_MNr_01.a_GATEWAY[2]
USINT #0—————_EthernetConfig_MNr_01.a_GATEWAY[3]

(* The configuration command *)

DWORD#16#12345600—_EthernetConfig_MNr_01.d_IP_CONFIG
```

Figure 26: Example: Assigning a variable IP address, gateway and subnet mask for the module

For the Ethernet module, this yields the IP address 133.12.195.3 + rotary switch and the subnet mask 255.255.192.0. The gateway has the address 0.0.0.0, i.e. no gateway is used.

#### d) Activating the TCP/IP configuration

The POU containing the elements for TCP/IP configuration has been created. After this, carry out the following steps:

- 1 Now link the POU in a cold boot and a warm restart task. After this, compile the program and load it as a boot project on the b maXX controller PLC.
- 2 Switch the b maXX system off and back on again, in order to download the boot project. The b maXX system must be switched on approximately 10 s, in order that the Ethernet module can accept the new configuration. However the configuration is not active. In order to activate this, a second switch off and back on again of the b maXX system is required.

The Ethernet module now has the TCP/IP configuration that you set. You can remove the program code from the created POU. The TCP/IP configuration is retained. In the same way, you can load another project (even a boot project) on the b maXX controller PLC without losing the TCP/IP configuration. You can only change the TCP/IP configuration of the Ethernet module by implementing again the code lines that were described above.

#### e) Checking the TCP/IP configuration

You can use PROPROGRAM wt II / ProProg wt III to check the settings of the Ethernet module's TCP/IP configuration.

To do this, proceed as follows:

- 1 Create a PROPROGRAM wt II / ProProg wt III project for the b maXX controller PLC if you have not already created a project for your application or by the Ethernet configuration.
- 2 PROPROGRAM wt II: In this project, link library BM\_TYPES\_20bd06 (or higher) if it is not already present.  
ProProg wt III: In this project, link library BM\_TYPES\_30bd01 (or higher) if it is not already present.
- 3 Create a global variable of data type ETHERNET\_PLC\_DIAG\_BMSTRUCT. You must connect this global variable to the base address to the Ethernet diagnostics. In this connection, the base address depends on the module number (01 to 05, set on S1). The following addresses are possible:

Module number (S1)	Base address for Ethernet diagnostics
01	%MB3.2012320
02	%MB3.3012320
03	%MB3.4012320
04	%MB3.5012320
05	%MB3.6012320
06 to 15	reserved

Example if the Ethernet option module has the module number 01 (S1 = 1):

PROPROGRAM wt II:

```
VAR_GLOBAL
_EthernetDiag_MNr_01 AT %MB3.2012320 : ETHERNET_PLC_DIAG_BMSTRUCT;
END_VAR
```

Figure 27: Example: Global variable for checking the TCP/IP configuration (PROPROGRAM wt II)

ProProg wt III:

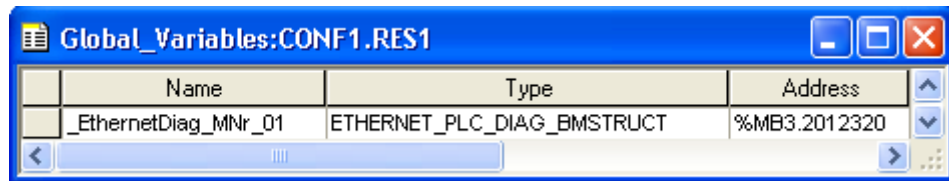


Figure 28: Example: Global variable for checking the TCP/IP configuration (ProProg wt III)

Where:

`_ EthernetDiag_MNr_01` is the variable name with data type short designation "\_" for STRUCT

`%MB3.2012320` is the base address for Ethernet diagnostics of the module with number 01.

`ETHERNET_PLC_DIAG_BMSTRUCT` is the data type of the variable

- 4 Structure elements `a_MAC_ADDRESS`, `a_IP_ADDRESS`, `a_IP_MASK`, `d_IP_CONFIG`, `d_DIP_SWITCH`, and `a_GATEWAY` of the created global variable are available for diagnostics. The individual elements have the following meanings:

Element	Meaning
<code>a_MAC_ADDRESS</code>	MAC address of the module. Field (array) containing 8 entries of data type BYTE in each case. Field indexes 0 to 5 are valid. Indexes 0 to 2 designate the code for manufacturer identification; indexes 3 to 5 designate the manufacturer code for the respective device.
<code>a_IP_ADDRESS</code> , <code>a_IP_MASK</code> , <code>a_GATEWAY</code>	Active IP address, subnet mask and gateway of the module. All three elements consist of one field (array) containing four entries each of data type USINT. Each address starts in field index 0 with the network section.
<code>d_IP_CONFIG</code>	Designates the type of evaluation of the IP address. 16#00000000 No evaluation 16#12345678 Fixed IP address 16#12345600 IP address + rotary switch S2 and S3
<code>d_DIP_SWITCH</code>	Settings of rotary switches S2 and S3 (0 to 31 = 0x00 to 0x1F, i.e. 0x1F is equivalent to S2 = 0x1 and S3 = 0xF)

- 5 Compile the program and load it as a project (even a boot project) on the b maXX controller PLC. You do not need to create a special POU.
- 6 Start the program and switch to online mode. Add the created global variable to TCP/IP diagnostics in the watch window; display the watch window and open the relevant structure elements. You can now see the set data.

Example:

Variable	Value	Default value	Type
[-] _EthernetDiag_MNr_1			ETHERNET_PLC_DIAG_BMSTRUCT
[-] a_MAC_ADDRESS			BYTE_8_BMARRAY
[0]	16#00		BYTE
[1]	16#02		BYTE
[2]	16#FB		BYTE
[3]	16#FF		BYTE
[4]	16#FE		BYTE
[5]	16#28		BYTE
[6]	16#00		BYTE
[7]	16#00		BYTE
[+] a_Reserved8			BYTE_8_BMARRAY
[-] a_IP_ADDRESS			USINT_4_BMARRAY
[0]	133		USINT
[1]	12		USINT
[2]	195		USINT
[3]	7		USINT
[-] a_IP_MASK			USINT_4_BMARRAY
[0]	255		USINT
[1]	255		USINT
[2]	192		USINT
[3]	0		USINT
d_IP_CONFIG	16#12345600		DWORD
d_GENERAL_STAT	16#00000001		DWORD
d_DIP_SWITCH	16#00000004		DWORD
[-] a_GATEWAY			USINT_4_BMARRAY
[0]	0		USINT
[1]	0		USINT
[2]	0		USINT
[3]	0		USINT

Figure 29: Example: Reading out the TCP/IP configuration in the watch window

### 3.4 Setting a connection via Ethernet-TCP/IP in PROPROG wt II / ProProg wt III

You set the uses of Ethernet-TCP/IP in PROPROG wt II / ProProg wt III individually for each resource in the user program. To do this, proceed as follows:

- In PROPROG wt II / ProProg wt III project open the dialog „Resource settings for SH03\_30“ via the context menu of the resource b maXX controller PLC
- Set the port to „DLL“

Setting the Ethernet communication source by choosing or stating the TCP/IP address

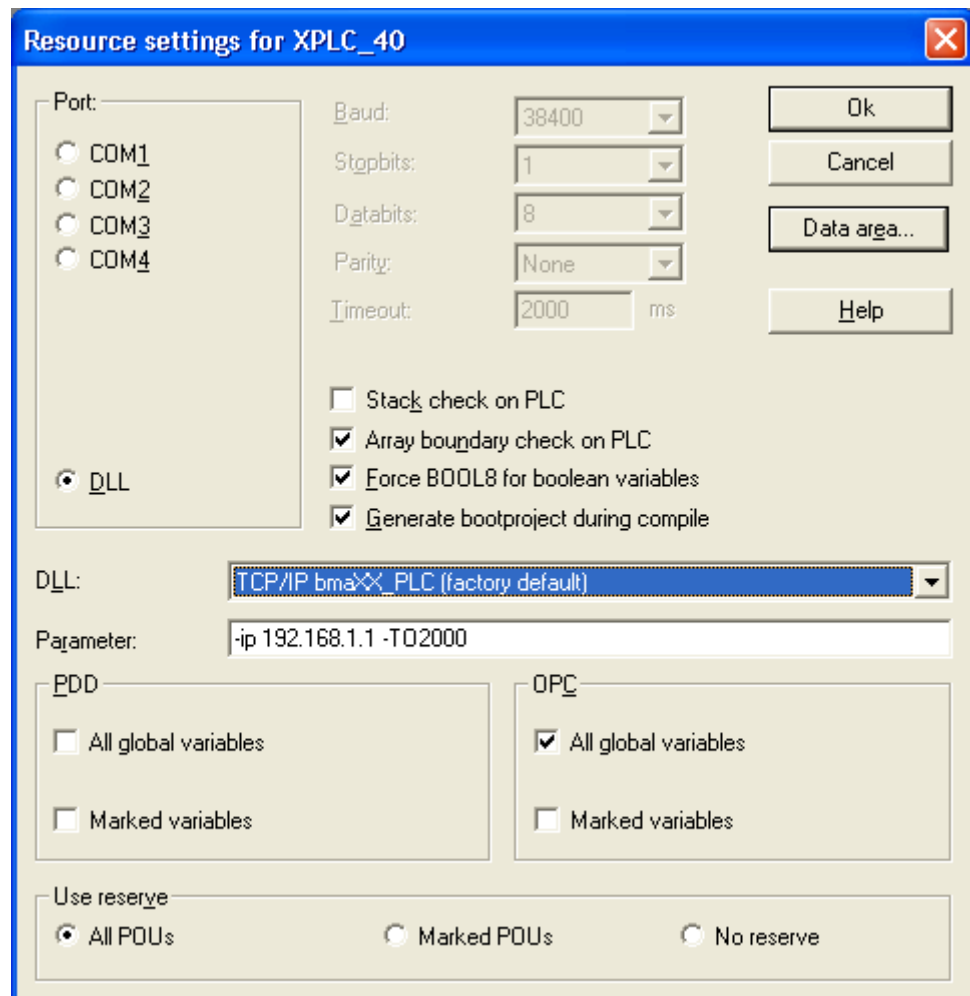


Figure 30: Setting TCP/IP in PROPROG wt II for one resource

After changing the port to "DLL", you can use the DLL menu to choose applications "Soft-PLC" (=TCP/IP local host (this PC)) and "b maXX PLC access via Ethernet" (=TCP/IP bmaXX\_PLC (factory default)):

- **Soft-PLC:**  
You do not need to change the preset TCP/IP address in field "Parameter" if the Soft-PLC is installed on the same computer. If installing Soft-PLC on another computer, you must enter here the appropriate TCP/IP target address (or the appropriate network name) to be able to access Soft-PLC via TCP/IP from PROPROG wt II / ProProg wt III.
- **b maXXcontroller PLC:**  
To be able to access the b maXX controller PLC via Ethernet, an Ethernet compliant module (e.g. BMC-M-ETH-02) must be fitted in the b maXX system in addition to the b maXX controller PLC. The TCP/IP address "192.168.1.1" that is preset in the parameter field corresponds to the IP address that is preinstalled on the Ethernet module when it leaves the factory, so that the user can contact the module first for the basic initialization to make the adjustments of its TCP/IP address for the TCP/IP network at the machine.



# 4

## CANOPEN

This chapter contains information on data exchange via CANopen. CANopen is only available with the module BMC-M-ETH-02 and BMC-M-CAN-04.

### 4.1 General information on CANopen and on use of CANopen master module

---

CANopen is a popular field-bus application layer based on the Controller Area Network (CAN) bus system and published by the international CAN organization CAN in Automation e.V. (CiA). The CANopen mechanisms and functionalities are described with various profiles. The communication profile defines the manner of data exchange and general specifications applicable to all CANopen devices. With the module CANopen master for b maXX controller PLC, CANopen and CANopen master functions according to the CiA communication profile DS 301 and CiA device profile DS 405, such as

- process data exchange via process data objects (PDOs) with high-priority identifiers
  - service data exchange via service data objects (SDOs) with low-priority identifiers
  - network management (NMT) for the implementation of network management functions, e.g. initialization, starting and resetting of network nodes.
  - synchronization (SYNC) for the synchronization of the real-time data exchange with PDOs
  - error treatment (EMERGENCY) for the detection of errors of a network node
  - network monitoring (NODE GUARDING) for the failure monitoring of network nodes
- can easily be realized.

The nine transfer rates 1 Mbit/s, 800 kbits/s, 500 kbits/s, 250 kbits/s, 125 kbits/s, 100 kbits/s, 50 kbits/s, 20 kbits/s and 10 kbits/s are available for the CAN. The CANopen master module can communicate with up to 32 CANopen network nodes (e.g. I/O modules and drives).

There are various methods available to you for achieving a data exchange with the CANopen master module via CANopen.

- ProMaster, ProCANopen, ProProg wt III and Motion Control

Simple, fast configuration of data exchange in ProMaster and ProCANopen. Simple, fast programming of the machine functions in ProProg wt III with the Motion Control function blocks and the network variables of the CANopen IO modules.

The evaluation of the network states and the network node states can also be carried out via function blocks from the library CANopen\_PLC01\_30bd00 (or higher).

For data exchange via CANopen with the module CANopen master and ProMaster, ProCANopen, ProProg wt III and Motion Control, the CANopen master module requires a software version of  $\geq$  **01.20** (i.e.  $\geq$  BMC-M-ETH-02/CAN-04-01-00-001-**001**).

Example projects:

ProMaster Project                      Example\_2\_1\_1.bmxml

IEC Project in ProProg wt III        Example\_BMC\_M\_CAN04\_MA\_1.mwt/.zwt

See [▶4.3 ProMaster, ProCANopen, ProProg wt III and Motion Control](#) from page 57 onward.

- PROPROG wt II

Programming of the data exchange and the machine functions in PROPROG wt II by using the IEC 61131-3 programming system PROPROG wt II with the library CANop405\_PLC01\_20bd02 (or higher)

Example project:

IEC Project in PROPROG wt II    CANopenMaster\_C\_Example.mwt/.zwt

See [▶4.4 Programming data exchange with PROPROG wt II and CANop405\\_PLC01\\_20bd03 library](#) from page 101 onward.

- PROPROG wt II, Motion Configurator and Motion Control

Simple, fast configuration of the data exchange in the Motion Configurator

Simple, fast programming of the machine functions in PROPROG wt II with the Motion Control function blocks.

See [▶4.5 CANopen and Motion Control with PROPROG wt II and motion configurator](#) from page 153 onward.

---

## 4.2 Basics of CAN and CANopen networks

---

### 4.2.1 Basics of CAN

---

A field bus system based on the CAN is carried out in the line structure. A three-wire cable with the connections CAN\_High, CAN\_Low and CAN\_Ground serves as the basis for data transfer. CAN uses a transfer balanced to ground to suppress common-mode interference. As a result, differential signals are evaluated.

#### Network

CAN is a multi-master network. Each node can access the bus actively and with the same authorization. CAN uses object-oriented addressing, i.e. the transmitted message is labeled with an identifier specified for the entire network. It represents the coded name of the message.

#### Bus access

Bus access is carried out with the CSMA/CA (Carrier Sense Multiple Access/Collision Avoidance) method. As each participant has the right, after detection of the necessary bus quiescence, to begin transmitting its message, collisions can occur. This is prevented by the bit-by-bit arbitration of the messages to be transmitted. Here two bus levels are differentiated, i.e. a dominant level, with a logical bit value of 0, and a recessive level, with a logical bit value of 1. In the worst case, all nodes ready to transmit begin with the transmission of their message on the bus at the same time. If a recessive bit of a node is overwritten by a dominant bit of another node, then the "recessive" node withdraws from the bus and attempts to transmit its message again following detection of the bus quiescence. This ensures that the most important, highest-priority message (with the lowest identifier) is transferred collision-free and without delay. For this reason it is, of course, necessary that each identifier may only be assigned once on the CAN bus.

#### Identifier

Various identifiers are available according to specification CAN 2.0A 2032. Each node can transmit without be requested to do so (multi-master capability). A transmitter transmits its message to all CAN nodes (broadcast), which then decide themselves based on the identifier whether or not they process the message further.

#### Error

Up to eight bytes of user data can be transmitted in a CAN data telegram. A CAN node can transmit error or overload telegrams to signal errors or overloading. This takes place on Layer 2 of the OSI/ISO reference model, the Data Link Layer, i.e. independently of the application. Due to high-quality error detection and treatment on Layer 2, a Hamming Distance (dimension of the error detection) of  $HD = 6$  is achieved, i.e. a maximum of five simultaneously occurring bit errors within a telegram are reliably detected as errors.

### 4.2.2 Basics of CANopen

CANopen is an open, and therefore manufacturer-independent field bus system based on the Layer 1 and 2 definitions of the CAN standard.

#### 4.2.2.1 CAL specification and profiles

With the application layer CAL a general description language for CAN networks was developed in the international CAN organization CAN in Automation e.V. (CiA). CAL provides a collection of communication services without exactly specifying their application. CANopen uses this description language. On the one hand, a subset of the communication services offered by CAL is used for the definition of an open communication interface. These definitions are contained in the communication profiles, whereby here the profile DS 301, in which the 'how' of communication is governed, is of particular importance. On the other hand, CANopen also specifies the meaning of the data in the respective device classes. These definitions are contained in the device profiles. For example, for I/Os this is the profile DS 401, and for drives the profile DSP 402. Elements in user interfaces programmable according to IEC 61131-3 are defined by the device profile DS 405. The CANopen master module for b maXX controller PLC supports the communication profile DS 301, DSP 302 (from FW 1.20.mit ProMaster) and the device profile DS 405. The communication profile DSP 302 is supported in conjunction with ProMaster from software version  $\geq$  **01.20** (i.e.  $\geq$  BMC-M-ETH-02/CAN-04-01-00-001-**001**).

#### 4.2.2.2 Communication and device-specific objects

For access via the network, the objects of a network node are always accessed. Objects assume various tasks. The data exchange via the CANopen field bus is configured with the communication profile of specific objects. As a device profile, specific objects are in direct contact to the device. The device functions can be used and changed via them. Objects are always addressed via an index (16-bit) and a sub-index (8-bit). Various areas are defined for objects depending on the meaning. The most important user areas are:

Index (hex)	Object
1000 - 1FFF	Area for the communication profile. Contains all objects required for communication, e.g. PDO, SDO.
2000 - 5FFF	Area for manufacturer-specific objects. Contains all objects not defined within a profile and that are manufacturer-dependent.
6000 - 9FFF	Area for the device profile. Contains the objects of the respectively supported device profile (DS 401, DS P402 etc.)

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

### 4.2.2.3 Data exchange and objects of physical bus system

From the standpoint of the communication relationships between CANopen nodes, the data exchange is described by the

- client-server model: each node is authorized by another node to request data or to transfer it to the other node

and

- Producer-consumer model: a node transmits data without being requested to do so; others listen along and can evaluate the data

A master-slave relationship (exactly one CANopen node organizes the data exchange) does not exist on the level of the communication relationships. A CANopen master is not understood to be a master on the communication level, but rather a master on the application level. Therefore, CANopen master functions mainly consist of network management tasks like

- starting and stopping network nodes
- monitoring network nodes
- configuration of the data exchange

In the process, the CANopen master makes use of the functions of the communication level. As functions of this kind are also closely linked to the actual application, it is practical that a CANopen master can be programmed under IEC 61131-3. With the CANopen master module, this is possible together with the library

- CANopen\_PLC01\_30bd00 (or higher) under ProProg wt III
- CANop405\_PLC01\_20bd03 (or higher) under PROPROG wt II

Data exchange in CANopen networks takes place as telegrams with which the user data are transmitted or with which the objects of a network node are accessed. Each telegram is marked by an identifier. The telegram types and the mechanisms for data exchange behind them are divided into groups:

- Process data objects (PDOs): Real-time data exchange with high-priority identifiers and up to 8 bytes per message.
  - ProMaster, ProCANopen, ProProg wt III (possibly the library CANopen\_PLC01\_30bd00 (or higher)): With the CANopen master module, a maximum of 256 PDOs can be written and 256 PDOs can be read simultaneously.
  - PROPROG wt II, the library CANop405\_PLC01\_20bd03 (or higher): With the CANopen master module, a maximum of 40 PDOs can be written and 63 PDOs can be read simultaneously.
- Service data objects (SDOs): Parameter data exchange with low-priority identifiers and data addressable by index/subindex.
  - ProMaster, ProCANopen, ProProg wt III (possibly the library CANopen\_PLC01\_30bd00 (or higher)): The CANopen master module supports the transfer types "expedited" (4 bytes per message) and "segmented" (up to 7 bytes per message). An SDO can be written or read.
  - PROPROG wt II, the library CANop405\_PLC01\_20bd03 (or higher): The transfer type for the CANopen master module is "expedited", i.e. up to 4 bytes can be transferred per message. Depending on the network configuration, up to eight different SDOs can be read or written simultaneously.

- Network management (NMT): Special object type for the implementation of network communication functions, e.g. initialization, starting and resetting of network nodes. The CANopen master module sends the commands as a broadcast to the respective network nodes.
- Synchronization (SYNC): Special object type for the synchronization of the real-time data exchange with PDOs. The CANopen master module sends the SYNC command as a broadcast to the respective network nodes.
- Error treatment (EMERGENCY): Special object type for the detection of errors of a network node. If an error occurs on a network node, an emergency telegram is transmitted by it, enabling this to be received and evaluated by the CANopen master module.
- Network monitoring
  - NODE GUARDING:
    - Special object type for the failure monitoring of network nodes. The CANopen master module can cyclically request the feedback from network nodes via a telegram to detect the failure of network nodes.
  - HEARTBEAT:
    - ProMaster, ProCANopen, ProProg wt III (possibly the library CANopen\_PLC01\_30bd00 (or higher)): Special object type for the failure monitoring of network nodes. The CANopen master module detects the failure of network nodes from the fact that these network nodes no longer transmit heartbeat telegrams. The advantage of the heartbeat compared to node guarding is reduced telegram traffic on the CANopen, and therefore a lower bus load.
    - PROPROG wt II, the library CANop405\_PLC01\_20bd03 (or higher): Heartbeat is not supported.



### 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 52, as a kind of data which can be accessed from outside or describes device properties. And on the other hand, it is also used to refer to the different telegram types and the mechanisms behind them for data exchange. In the second case, these objects can be seen as objects of the physical CANopen bus system, however not as objects which can be accessed via indices and sub-indices. These objects of the physical CANopen bus system transport the communication and device-specific objects.

In contrast to other field bus systems, these specifications also result in the telegrams being identified by the individual nodes themselves. Each network node decides itself when it wants to transmit data and also decides itself which telegrams it evaluates. However, it is also possible via so-called remote telegrams to request that other network nodes transmit data.

#### 4.2.2.4 Predefined identifiers

To establish peer-to-peer communication between the master and the other network nodes directly following a boot-up, a predefined identifier is assigned for the telegrams.

This identifier assignment can be reconfigured by the user. An identifier (according to the CAN definition) is divided into a function code and a 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 equivalent to the node number in the network. For the seven bits for the module ID, a theoretical maximum number of 127 nodes results for each CANopen network. For physical reasons, a maximum of 32 nodes can be operated in a network with the CANopen master module. A total of 11 bits results in the COB-ID (Communication Object Identifier). The following objects are predefined:

Broadcast objects (module ID = 0):

Object name	Binary function code	Resulting decimal COB-ID	Resulting hexadecimal COB-ID
NMT	0000	0	16#0
SYNC	0001	128	16#80
TIME STAMP <sup>1)</sup>	0010	256	16#100

<sup>1)</sup> The time stamp object is not supported by the CANopen master module.

Peer-to-peer objects:

Object name	Binary function code	Resulting decimal COB-ID	Resulting hexadecimal COB-ID
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 und RX for receive. Please note that the transmission direction is seen from the network nodes and not from the master. This means that if, for example, a PDO is transmitted from the module, then this is an RxPDO for the evaluating network

node and must therefore be transferred with an RX identifier on the CANopen bus. The same applies to the module ID. This must also be considered from the standpoint of network node and not of the master, i.e. the COB-ID contains the module ID of the network node and not of the master.

However, these conditions only apply to the predefined identifier assignment. According to the communication profile DS 301, the COB-IDs of the CANopen telegrams can be specified as desired by the user for each network node via objects specific to the communication profile. This makes it possible, for example, to also establish process data communication directly between two (or more) network nodes.

### 4.2.2.5 Telegram structure and priority

The most important CANopen elements in the CAN telegram are the COB-ID, the remote bit, the data length and the data. As the COB-ID is equivalent to the CAN identifier, low-value COB-IDs have a higher priority due to the CAN CSMA/CA transmission process (see the chapter [Basics of CAN](#) on Page 51). The predefined identifier assignment therefore results in the fact that NMT telegrams have the highest priority, followed by SYNC telegrams. As the COB-ID also contains the network node number, telegrams from network nodes with a lower node number also have a higher priority.

The most important elements in the CANopen telegram 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 the byte is dependent on the telegram type. For SDOs with "expedited" transfer, this is a maximum of 4 bytes, as the other 4 bytes are required for the identification of the data type. For SDOs with "segmented" transfer, this is a maximum of 7 bytes; 1 byte is required to control the data transfer. PDOs can use all 8 bytes. SYNC telegrams have no data content.



### 4.3 ProMaster, ProCANopen, ProProg wt III and Motion Control

In this chapter we explain how you can easily and quickly put a CANopen network into operation with the b maXX controller PLC and the CANopen master module. This chapter is accompanied by a ProMaster example project.

A CANopen network consists of a CANopen master and one or several CANopen slaves.

The machine configuration is created from these devices with the Baumüller ProMaster engineering framework. The CANopen communication between the individual devices is configured with the ProCANopen field bus configurator. By using Motion Control, the configuration settings are reduced to a minimum and the user can concentrate completely on the actual application (machine functionality) in the IEC 61131-3 programming system ProProg wt III.

The data and settings for CANopen which result during the configuration of CANopen communication are stored on the CANopen master module. This concerns both the data for the CANopen master itself and the data for the CANopen slaves. The CANopen slaves do not receive their data from the CANopen master until after the machine is switched on.

The data and settings for ProProg wt III, which are stored during the configuration of CANopen communication, are stored in the IEC project (ProProg wt III project) of the device with the CANopen master module. This IEC project is loaded on the b maXX controller PLC, to which the CANopen master is connected.

The Ethernet module with CANopen master (BMC-M-ETH-02) enables Ethernet communication with Single Axis Motion Control. Ethernet communication should not be used with Multi Axis Motion Control, as this reduces the synchronization accuracy in the CANopen network.

---

#### NOTE



Do not use Ethernet communication and Multi Axis Motion Control together on an Ethernet module with CANopen master (BMC-M-ETH-02).

---

No "not Multi Axis Motion Control" CANopen slaves should be used in a CANopen network in which the Multi Axis Motion Control is used. This especially applies to IO modules. The "not Multi Axis Motion Control" CANopen slaves can interfere with the synchronicity in the CANopen network. Use a second CANopen master module for "not Multi Axis Motion Control" CANopen slaves.

---

#### NOTE



Do not use "not Multi Axis Motion Control" CANopen slaves and "Multi Axis Motion Control" CANopen slaves together on one (Ethernet with) CANopen master module (BMC-M-ETH-02 / BMC-M-CAN-04).

### 4.3.1 Requirements

---

The requirements are

- PC with
  - Engineering Framework ProMaster
  - IEC 61131-3 programming system ProProg wt III with the firmware libraries
    - Bit\_UTIL\_30bd00
    - OmegaOS\_30bd00
    - SYSTEM2\_C\_PLC01\_30bd00
    - MC\_SYS\_30bd00
  - and the libraries
    - BM\_TYPES\_30bd00
    - SYSTEM1\_C\_PLC01\_30bd00
    - CANopen\_PLC01\_30bd00 (for devices that do not support Motion Control)
    - MOTION\_TYPES\_30bd00
    - MOTION\_CONTROL\_30bd00
    - MOTION\_MULTI\_AXIS\_30bd00 (if Multi Axis Motion Control is used)
  - Drive control program for b maXX WinBASS II (for commissioning the CANopen (controller) slaves).
- Physical commissioning of the components of the CANopen network; in our example, these are the CANopen master
  - b maXX controller PLC (BMC-M-PLC-0x-...) with
    - CANopen master module (BMC-M-CAN-04-... or BMC-M-ETH-02-...)
  - and the CANopen slave
    - b maXX 4400 with
      - CANopen slave option module (BM4-O-CAN-03-...)

See the respective operating instructions for more information on this topic.

### 4.3.2 Steps to be carried out

---

To use the CANopen master module for b maXX controller PLC for Module Control in a CANopen network, the following steps must be carried out.

- 1 Commissioning of the CANopen network
- 2 Creation of an IEC 61131-3 project for ProProg wt III with a Motion Control template
- 3 Creation of a machine configuration in ProMaster
- 4 Configuration of the CANopen communication with ProCANopen and subsequent downloading to the CANopen master (testing of the CANopen network with this configuration if necessary)
- 5 Configuration of the PLC with connection of the IEC project and the ProMaster, configuration of the cam disk data and configuration of the IOs if necessary
- 6 Programming of the IEC 61131-3 projects for ProProg wt III (application) and subsequent downloading to the b maXX controller PLC if necessary
- 7 Running of the application in the CANopen network

Of course, you can also begin with step 2 and wait until step to carry out the physical commissioning (step 1).

In the process, the ProMaster project **Example\_2\_1\_1.bmxml** and the IEC 61131-3 project (ProProg wt III project) **Example\_BMC\_M\_CAN04\_MA\_1.mwt/.zwt** are created.

### 4.3.3 Commissioning of the CANopen network

For details on commissioning the CANopen network, please see the operating instructions on the BMC-M-ETH-02/BMC-M-CAN-04 module and the operating instructions of the other CANopen network nodes.

Following the physical commissioning of the network and the connection of the power supply of the b maXX system (also see the operating instructions of the power supply unit for the b maXX controller PLC), the CANopen master module is ready for operation after approx. 5 seconds.

H1 (green): Flashes H2 (red): Off	CANopen: The module waits for the initialization by an application program on the b maXX controller PLC
H1 (green): Off H2 (red): On	CANopen: The CAN is in the Bus-Off state

Flashes: ton = 200 ms, toff = 200 ms

All other states of the LEDs H1 and H2 indicate errors. For more information on this topic, please see the related operating instructions for the CANopen master module BMC-M-ETH-02/BMC-M-CAN-04.

### 4.3.4 Creating a IEC project with ProProg wt III

In the following we will explain how a Motion Control IEC project is created.

Multi Axis Motion Control is set in the ProMaster default settings for Motion Control. Therefore, it is practical to first create an IEC project for Multi Axis Motion Control.

Open ProProg wt III and create a new project by selecting an IEC template with the "File\New Project" menu.

For the example we use the IEC template project „Tmpl\_C\_PLC01\_MA\_1\_0103.zwt“ (for BMC\_M\_PLC01). You will find this project in the ProMaster installation directory \\Baumueller\ProMaster.NET in the subdirectory \\...\projects\IEC-Templates. Unpack the IEC template „Tmpl\_C\_PLC01\_MA\_1\_0103.zwt“ as „Example\_BMC\_M\_CAN04\_MA\_1.mwt“.

The template contains the libraries

- Bit\_UTIL\_30bd00
- BM\_TYPES\_30bd00
- MC\_SYS\_30bd00
- MOTION\_TYPES\_30bd00
- MOTION\_CONTROL\_30bd00
- MOTION\_MULTI\_AXIS\_30bd00

If you do not want to use all axes as Motion Control axes, or if you want to integrate other CANopen devices in your machine configuration, you must now also integrate the libraries

## 4.3 ProMaster, ProCANopen, ProProg wt III and Motion Control

- BM\_TYPES\_30bd00
- CANopen\_PLC01\_30bd00

for the evaluation of the network states and the network node states in your project.

Save the project. In our example we use the project name "Example\_BMC\_M\_CAN04\_MA\_1.mwt".

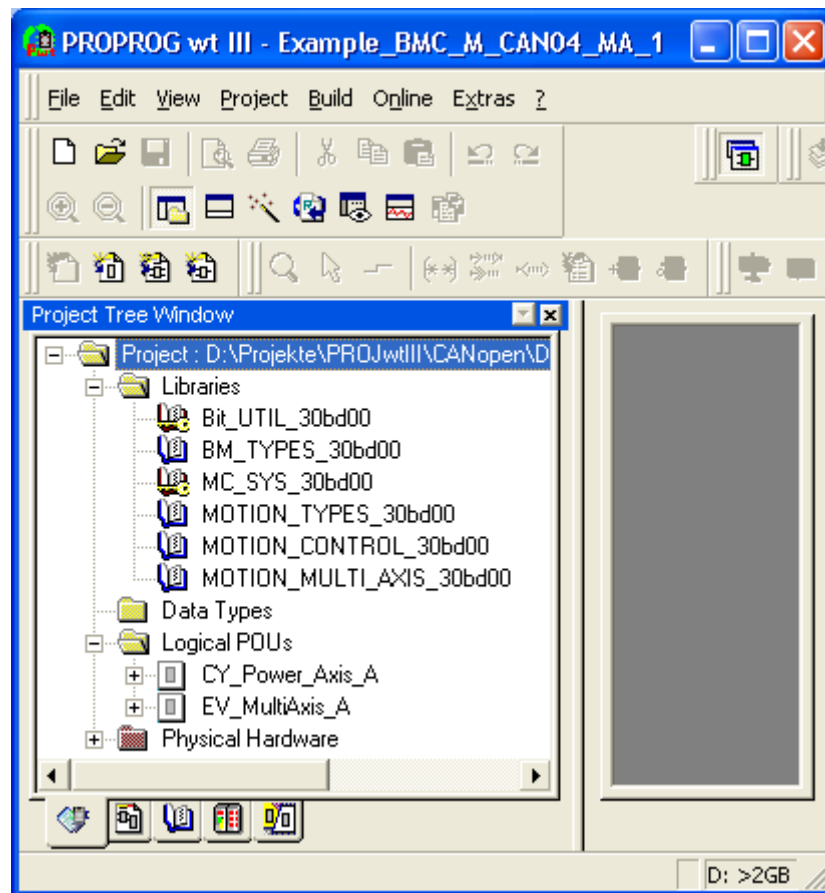


Figure 31: ProProg wt III - Creating the project "Example\_BMC\_M\_CAN04\_MA\_1.mwt"

Now the communication settings must also be configured. The serial port COM1 is the default setting. If you want to use the Ethernet, see the chapter [▶3 Ethernet◀](#) from page 15 onward.

For the use of an IEC project in ProMaster, it is absolutely necessary that the checkbox "Create book project during compiling" be activated for ProProg wt III in the "Resource - Settings" window (Mark resource and select "Settings" context menu). This generates the file "bootfile.pro", which is required for downloading onto the b maXX controller PLC.



### NOTE

In ProProg wt III the checkbox "Create boot project during compiling" must be activated under "Resource - Settings".

The "Example\_BMC\_M\_CAN04\_MA\_1.mwt" project is connected later to the ProMaster Project in the chapter [▶4.3.7 Configuring PLC◀](#) from page 82 onward.

Now close ProProg wt III via the "File\Exit" menu.

#### 4.3.5 Creation of a machine configuration in ProMaster

In this section we explain how a project is created in ProMaster and how a machine configuration is created.

Open ProMaster.

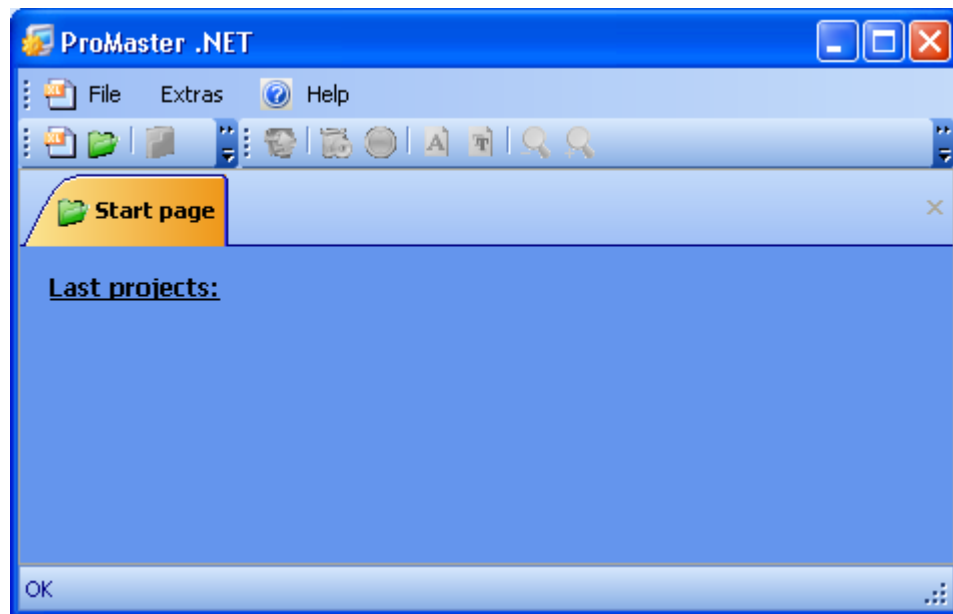


Figure 32: ProMaster without projects

Create a new ProMaster project by opening the "Project Settings" window with the "File\New Project" menu.

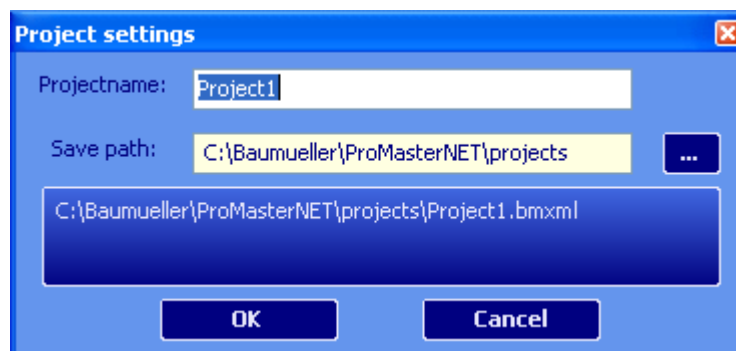


Figure 33: ProMaster - Project name

Specify the name (in the edit box) and the storage location (via the path selection) of the project. We'll use the name "Example\_2\_1\_1.bmxml" for our project.

## 4.3 ProMaster, ProCANopen, ProProg wt III and Motion Control

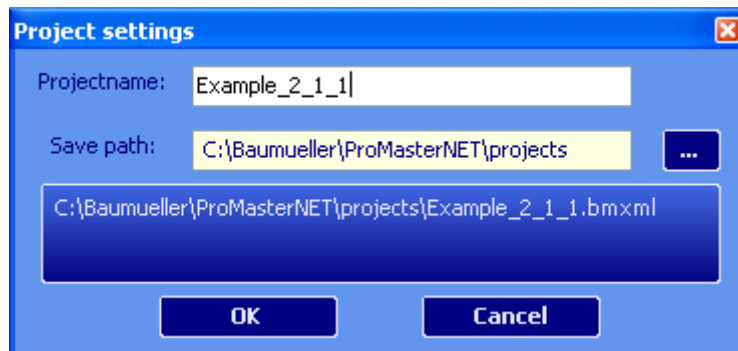


Figure 34: ProMaster - Assign project name

Apply the settings from the "Project Settings" window by clicking the "OK" button. ProMaster now changes to the network view.

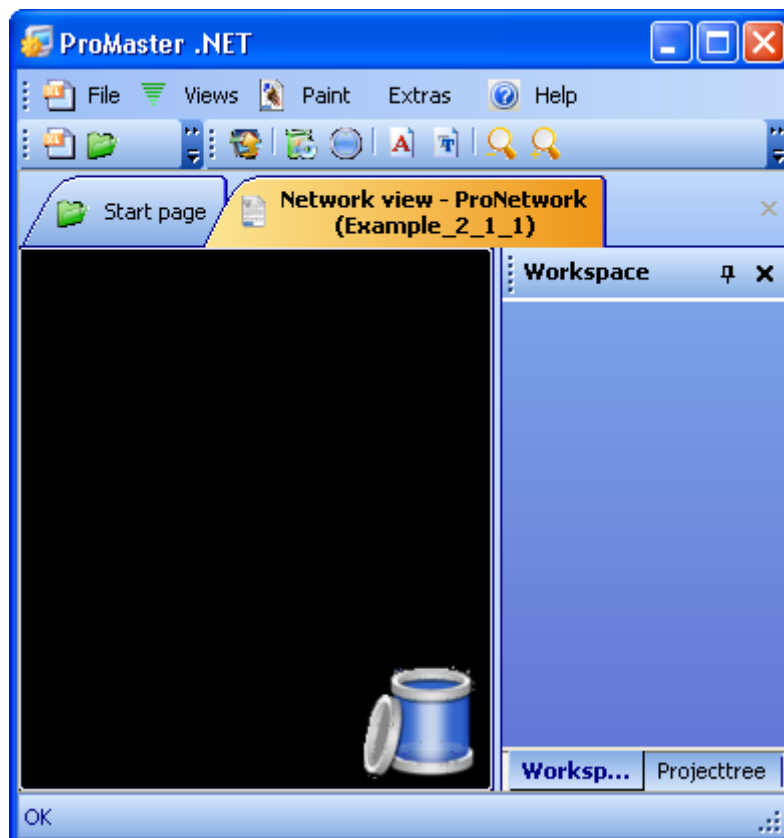


Figure 35: ProMaster - Assign project name

Now open the Baumüller catalog via the "Views\Catalog" menu. The Baumüller catalog contains the devices, bus system and components Baumüller provides you with in the default configuration.

For our example we use

- the devices

- "b maXX controller PLC CANopen master" (from the group "b maXX controller PLC System")
- "b maXX 4000 CANopen slave (2-row)" (from the group "b maXX 4000 Drive"; b maXX 4000 with two slots, one of which contains the CANopen slave option module)
- the bus system
  - "bus\_canopen" (CANopen), is automatically created.

Drag the device "bmaXX controller PLC CANopen master" out of the Baumüller catalog and into the ProMaster field view with the Drag&Drop function. The bus system has been connected automatically. Now drag the device "bmaXX 4000 CANopen slave (2-row)" onto the CANopen bus with the Drag&Drop function (dropping is not possible until the bus changes color).

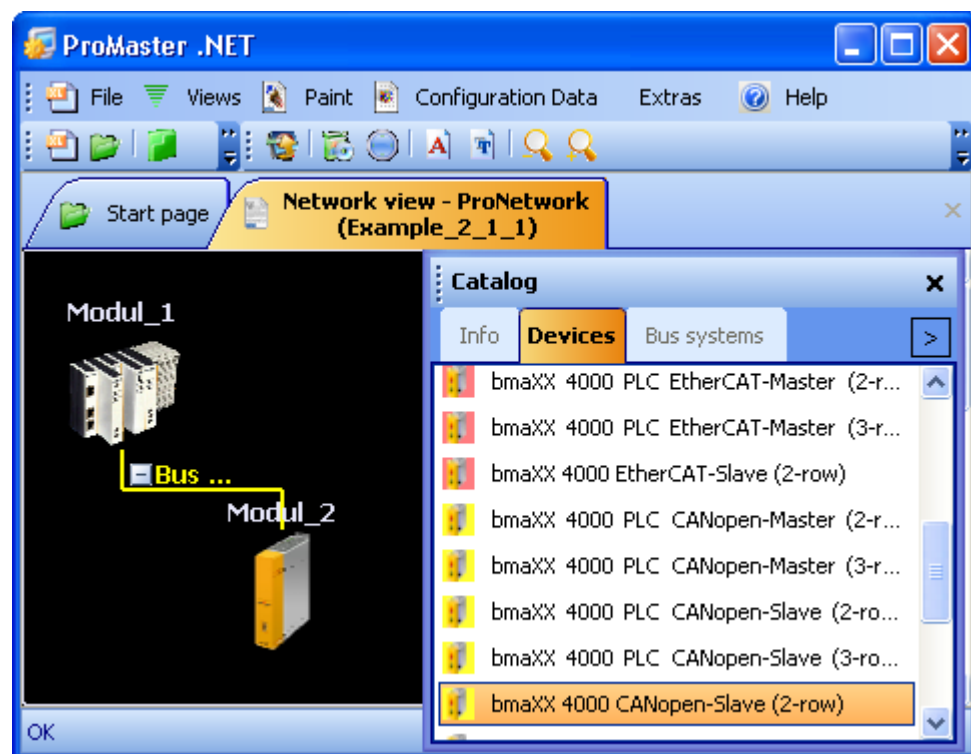


Figure 36: ProMaster - Machine configuration created

In the picture you now see the CANopen master (Module 1) and the CANopen slave (Module 2). Additional information on the modules is available via the respective tooltip or the "Project Tree Window". The "Catalog" window is closed for improved clarity.

The names (Module 1 and Module 2) are changed by clicking on the respective module and then opening the properties window of the respective module via the context menu and "Properties".

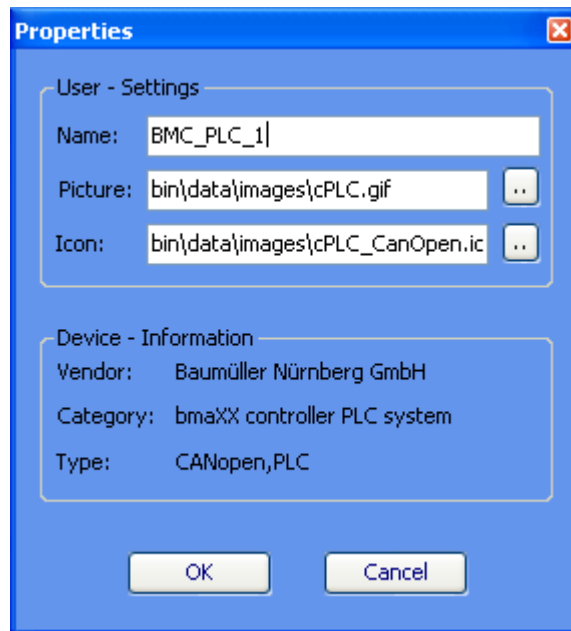


Figure 37: ProMaster - Module properties

For our example we assign the CANopen master the name BMC\_PLC\_1, and for the CANopen slave the name Namen \_Axis\_A (\_Axis\_A is the default axis variable name of the Motion Control axis in our IEC project). We keep the respective images (\*.bmp). The new names now appear in the ProMaster network view and in the project tree.



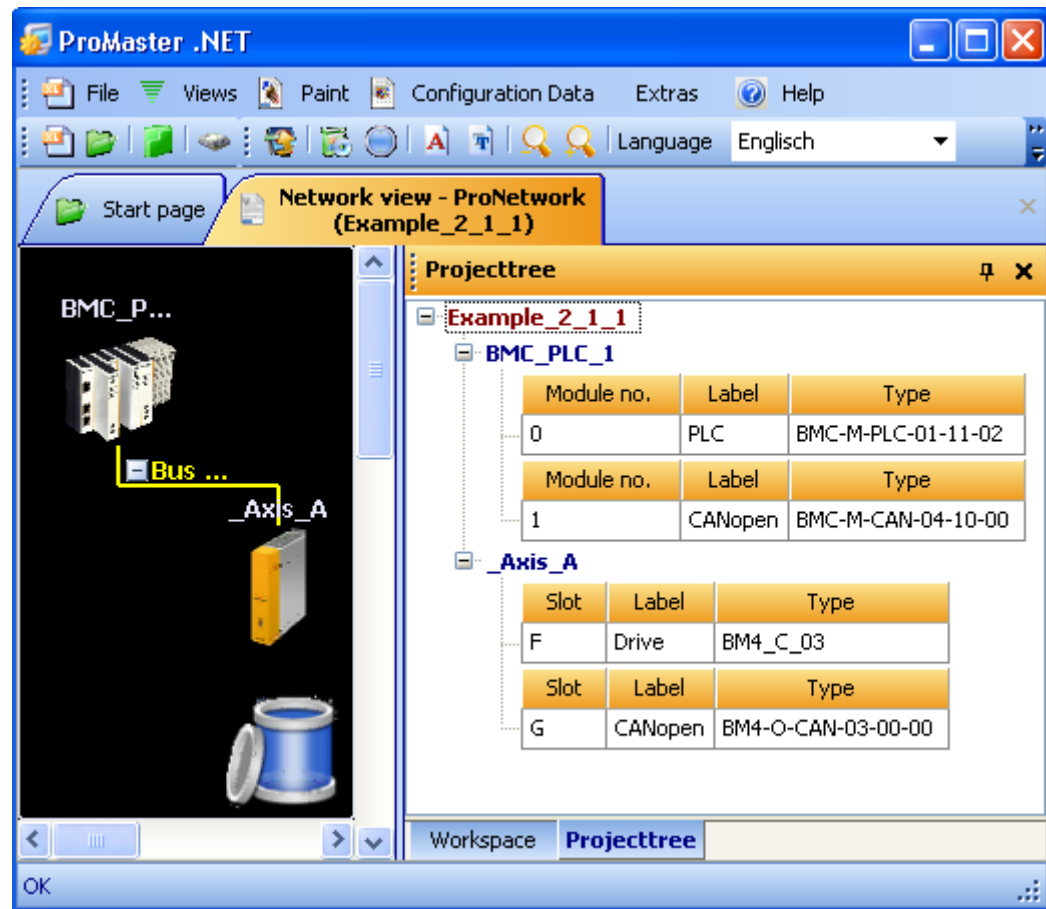


Figure 38: ProMaster - Machine configuration - Modules renamed

You can set the communication settings for the communication of ProMaster to the b maXX controller PLC by clicking on the device "BMC\_PL\_C\_1" and selecting "Communication Settings" via the context menu. Then save the project with the "File\Save Project" menu.

In ProMaster, Motion Control is activated for the components with CANopen (CANopen master, CANopen slave drive) in the default configuration. This reduces the settings to be configured to a minimum.

If you want to use the default settings (e.g. CANopen baud rate 1 MBit/s and the automatically assigned CANopen node-ID), you do not need to carry out any configuration of the CANopen communication with ProCANopen.

In this case, you continue with [▶4.3.7 Configuring PLC◀](#) from page 82 onward.

#### 4.3.6 Configuring CANopen communication with ProCANopen

In ProMaster, Motion Control is activated for the components with CANopen (CANopen master, CANopen slave drive) in the default configuration. This reduces the settings to be configured to a minimum.

If you want to use the default settings (e.g. CANopen baud rate 1 MBit/s and the automatically assigned CANopen node-ID), you do not need to carry out any configuration of the CANopen communication with ProCANopen.

In this case, you continue with [▶4.3.7 Configuring PLC◀](#) from page 82 onward.

We explain how you can change the settings of the CANopen communication in the following section.

First, the CANopen slaves are configured, then the CANopen master is configured.

### 4.3.6.1 Configuring CANopen slave communication

---

The CANopen slave communication is configured for each CANopen slave individually.

Open ProCANopen for our CANopen slave by clicking on the device "\_Axis\_A" and then select "Configuration Data (Components)\CANopen Slave (Slot G)\Configure Slave Bus (ProCANopen)" via the context menu.

The CANopen slave was already set with the default setting when creating the machine configuration so that the CANopen settings for Motion Control are configured (Drag&Drop of the slave on a master for which the Motion Control is activated).

The window for the CANopen slave communication configuration is opened.

#### Ident tab

For Motion Control, you can set the slot of the CANopen slave option module in the b maXX 4400, the name of this module (device name) in ProMaster and the baud rate and the node-ID for CANopen in the "Ident" tab. The baud rate and node-ID were applied from the CANopen configuration of the CANopen master module.

Changing the device name, baud rate and node-ID affects the ProMaster project.

---

#### NOTE



Reducing the baud rate with an unchanged "Cycle time for SYNC" (Network tab) can result in an excessively high bus load on the CANopen and negatively affect the synchronicity.

---

Changing the device name also has a negative effect on the Motion Control axis name in the IEC 61131-3 project in ProProg wt III.

---

#### NOTE



Changes to the device name also have an effect on the Motion Control axis name in the global variable worksheet in the IEC 61131-3 project in ProProg wt III.

This change is **not** automatically carried out in the local variable worksheets and in the code worksheets in the IEC 61131-3 project in ProProg wt III.

This remains the explicit task of the user (for safety reasons).

---

In addition, CANopen-specific information like the device type, CANopen profile, manufacturer-ID etc. are also displayed.

With series production of machines, it can be checked with the "Check during boot-up" checkboxes during start-up, whether the correct devices are connected.

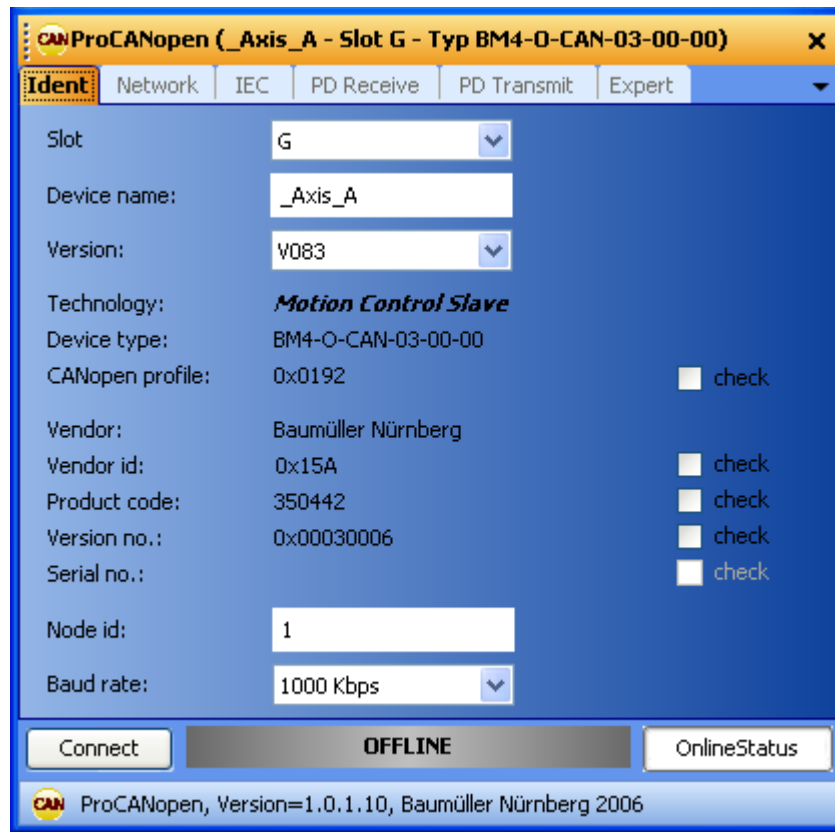


Figure 39: ProCANopen - CANopen slave communication configuration Ident tab

### Network tab

You can set the SYNC cycle time and guarding or heartbeat in the "Network" tab.

Changing the SYNC cycle time and the guarding or heartbeat settings affects the Pro-Master project. The SYNC cycle time is the time interval in which the CANopen master transmits the SYNC telegram on the CANopen bus.

#### NOTE



A CANopen slave can only support guarding or heartbeat.

#### NOTE



Reducing the "Cycle time for SYNC" with an unchanged "Baud rate" (Ident tab) can result in an excessively high bus load on the CANopen, and therefore negatively affect communication and synchronicity.

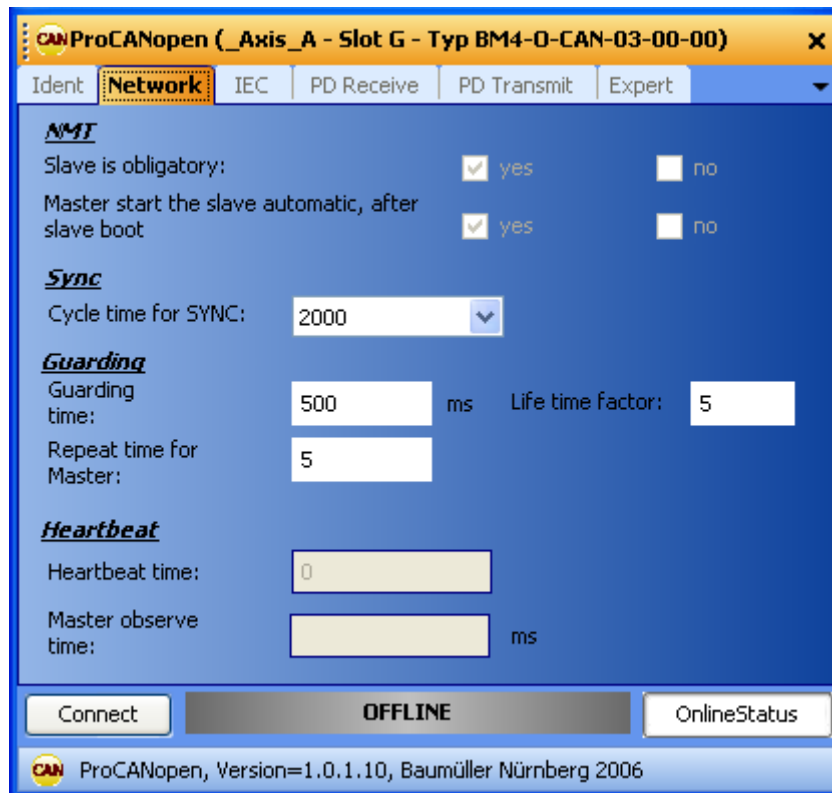


Figure 40: ProCANopen - CANopen slave communication configuration Network tab

### IEC tab

During the configuration of the CANopen slave communication, the CANopen slave objects (via CANopen master objects) are assigned to the network variables for the IEC project on the PLC.

The variables shown in the IEC tab are also called network variables. The process data are written and read in the IEC project via the network variables.

When using Motion Control, some of the assignments are created with the default settings.

The network variables and their data type in the IEC project on the PLC are shown in the "IEC" register.

The calculation of the (IEC) address of the network variables is not carried out until after "Update list" in the chapter [Download tab](#) in Chapter "4.3.6.2 Configuring CANopen master communication" from page 80 onward.

Other network variables are configured in the PD Receive tab and the PD Transmit tab.

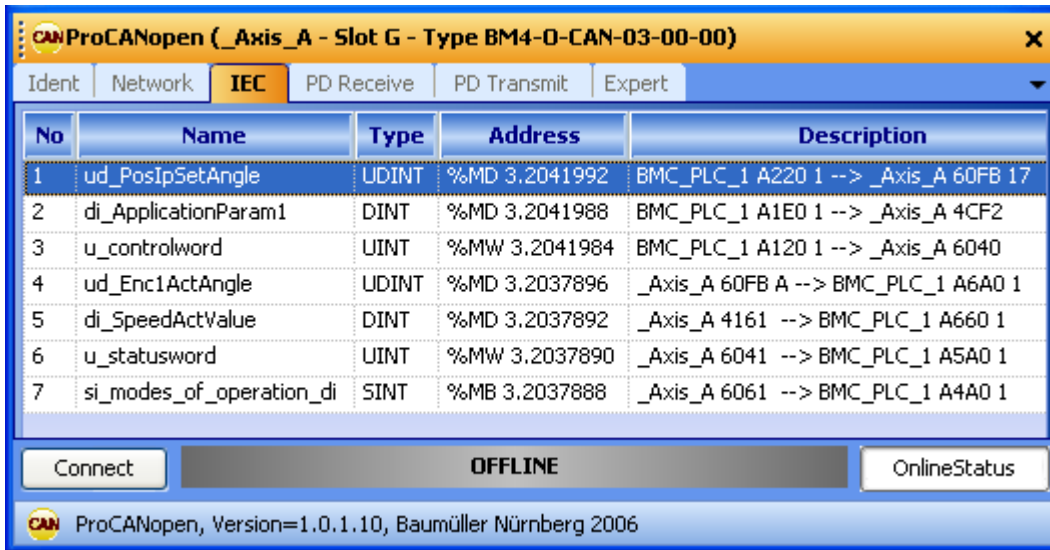


Figure 41: ProCANopen - CANopen slave communication configuration IEC tab following configuration of the CANopen slaves and "Update list" (CANopen master communication configuration Download tab)

### PD Receive tab

The communication from the master to the slave is configured in the "PD Receive" tab.

The default configuration is already set for Motion Control. You can add to the configuration. When doing so, note that the Motion Control configuration has a higher priority than the configuration by the user.

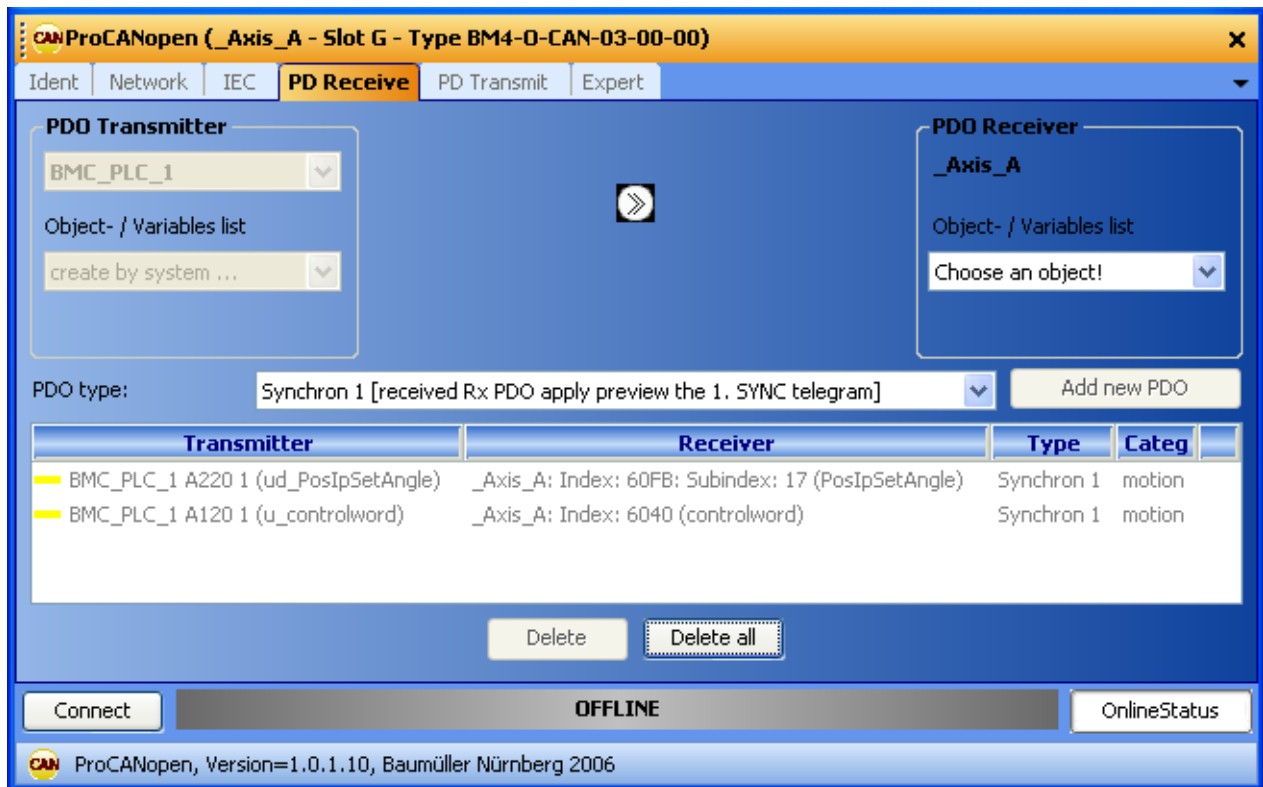


Figure 42: ProCANopen - Motion Control CANopen slave communication configuration "PD Receive" tab with default Motion Control configuration

The PDO transmitter is the CANopen master with Motion Control.

The linking of an object in the CANopen slave to an object in the CANopen master takes place here. This object in the CANopen master is written from within the IEC program via a network variable.

Procedure:

- Selection of an object of the CANopen slave in the "PDO Receiver" area in the "Object/Variable List" combination box.

Example: We select CANopen slave object 0x4CF2 (Application parameter 1) in the "PDO Receiver" area in the "Object/Variable List" combination box.

- Selection of the PDO transmission type from the "PDO Transmission Type" combination box. You can choose between Synchronous 1 to Synchronous 240 and Asynchronous 254 and Asynchronous 255.

Example: For our example we select "Synchronous 1", i.e. the Receive PDO (or its content) received before the last SYNC telegram is applied when the new SYNC telegram has been received.

- The link just set with the "Add new PDO" button is checked for conformance with the PDO entries and entered in the list.

The CANopen slave object 0x4CF2 (Application parameter 1) is now linked to the CANopen master object 0xA1E0, subindex 0x01. ProMaster and ProCANopen automatically generate an IEC variable (network variable) for the IEC project for this CANopen master object. In our case the name of the network variable in the IEC project is "di\_ApplicationParam1". In the IEC project this is written to this network variable. The

written value is written to the CANopen slave object 0x4CF2 via the CANopen master object 0xA1E0 - 0x01. See [▶IEC tab◀](#) in Chapter "4.3.6.2 Configuring CANopen master communication" from page 79 onward.

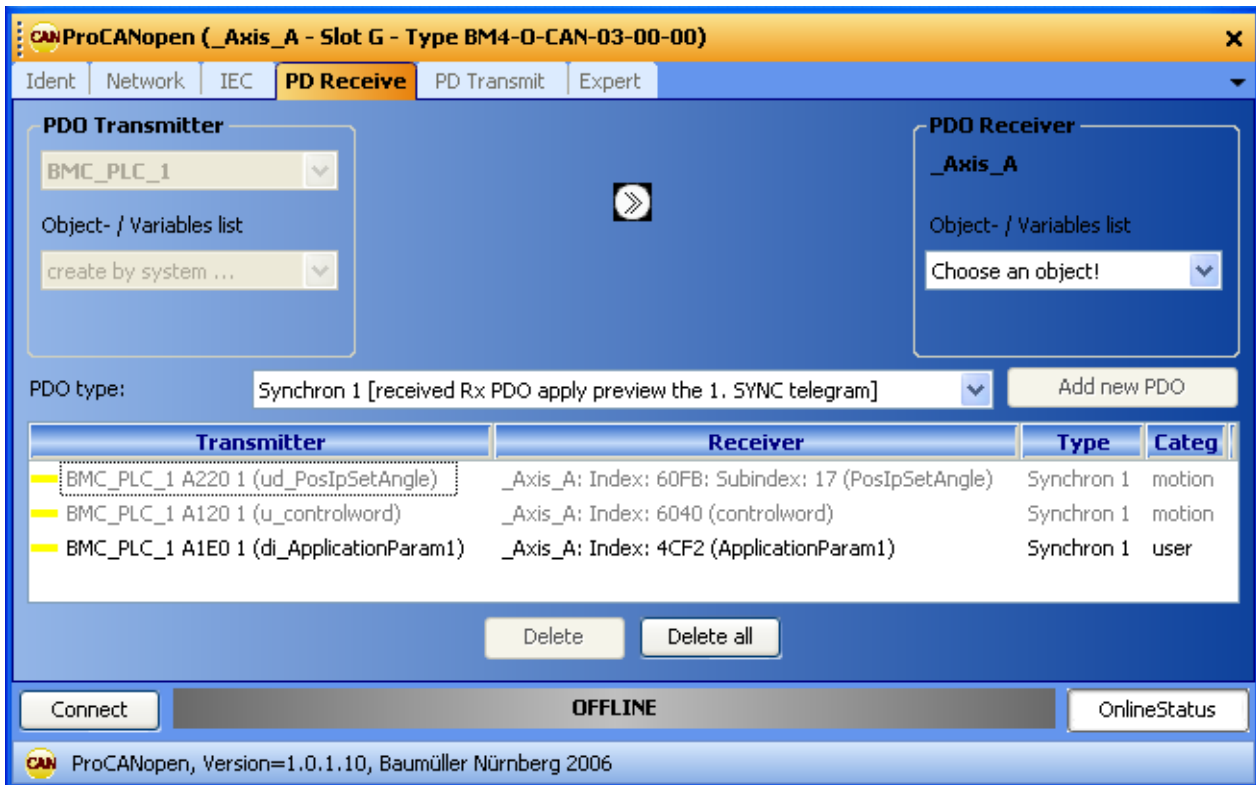


Figure 43: ProCANopen - Motion Control CANopen slave communication configuration "PD Receive" tab with default Motion Control configuration and additional object

#### NOTE



ProMaster automatically takes the limit of 30 characters for a variable name in the IEC project in ProProg wt III into account.

The name of the variables from the additional configuration can be changed later during the configuration of the PLC, in the chapter [▶4.3.7.5 CANopen◀](#) from page 93 onward.

#### PD Transmit tab

The communication from the slave to the master is configured in the "PD Transmit" tab. The default configuration is already set for Motion Control. You can carry out an additional configuration. When doing so, note that the Motion Control configuration has a higher priority than the configuration by the user.

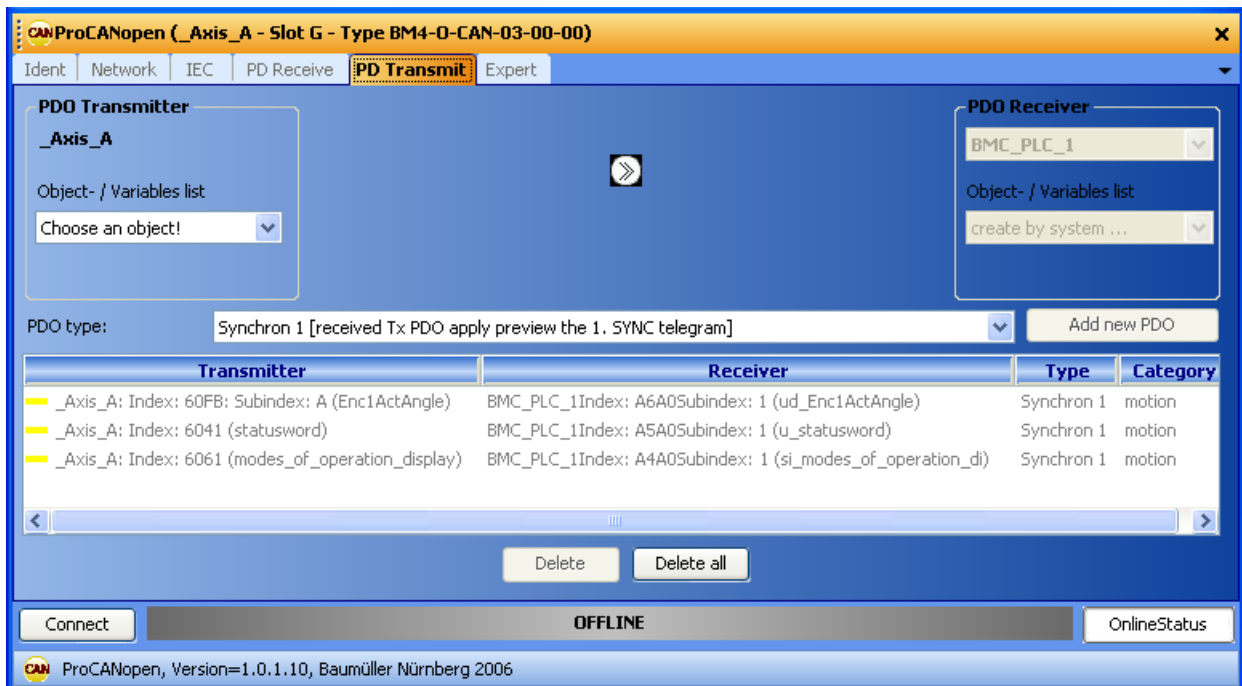


Figure 44: ProCANopen - Motion Control CANopen slave communication configuration "PD Transmit" tab with default Motion Control configuration

The PDO receiver is the CANopen master with Motion Control.

The linking of an object in the CANopen slave to an object in the CANopen master takes place here. This object in the CANopen master is read via a network variable in the IEC program.

Procedure:

- Selection of the object to be transmitted in the "PDO Transmitter" area in the "Object/Variable List" combination box.

Example: We select CANopen slave object 0x4161 (Actual speed value) in the "PDO Transmitter" area in the "Object/Variable List" combination box.

- Selection of the PDO transmission type from the "PDO Transmission Type" combination box. You can choose between Synchronous 1 to Synchronous 240 and Asynchronous 254 and Asynchronous 255.

Example: We select "Synchronous 1", i.e. the transmission is carried out in each case after the reception of the SYNC telegram (transmission after the first SYNC telegram since the last transmission).

- The link just set with the "Add new PDO" button is checked for conformance with the PDO entries and entered in the list.

The CANopen slave object 0x4161 (Actual speed value) is now linked to the CANopen master object 0xA660, subindex 0x01. ProMaster and ProCANopen automatically generate an IEC variable (network variable) for the IEC project for this CANopen master object. In our case the name of the network variable in the IEC project is "di\_SpeedActValue". In the IEC project this network variable is read. The value is read from the CANopen slave object 0x4161 via the CANopen master object 0xA660 - 0x01. See [► IEC tab](#) in Chapter "4.3.6.2 Configuring CANopen master communication" from page 79 onward.



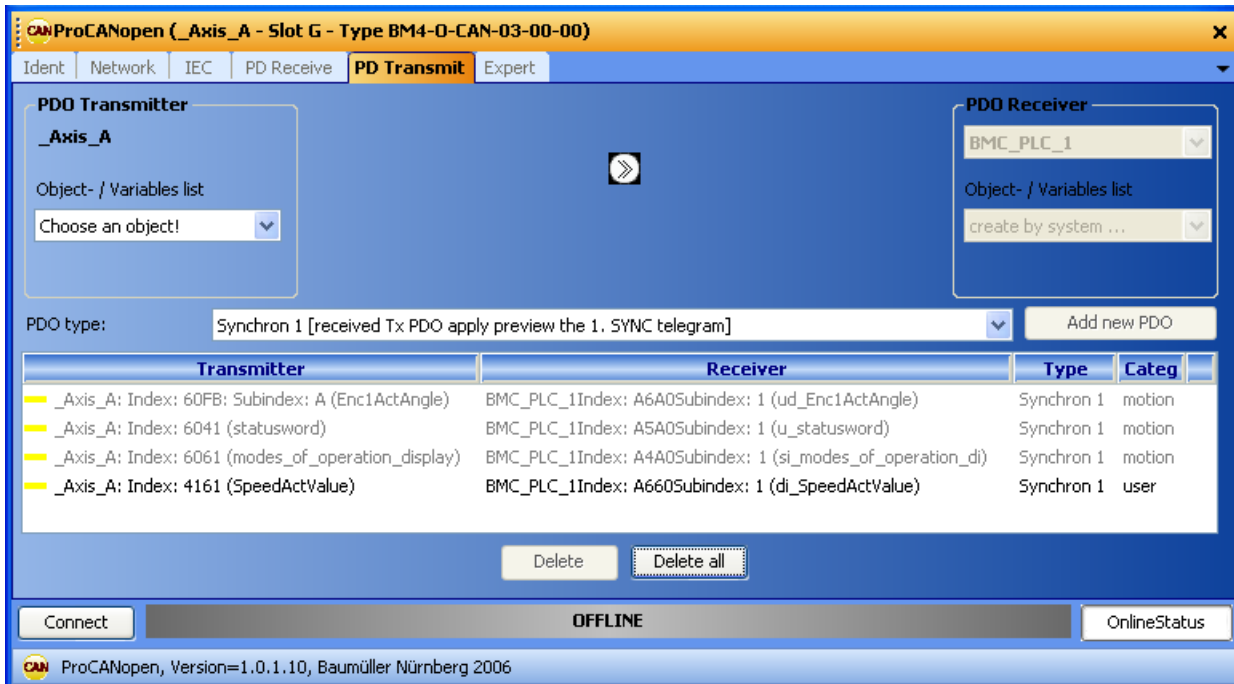


Figure 45: ProCANopen - Motion Control CANopen slave communication configuration "PD Transmit" tab with default Motion Control configuration and additional object



#### NOTE

ProMaster automatically takes the limit of 30 characters for a variable name in the IEC project in ProProg wt III into account.

The name of the variables from the additional configuration can be changed later during the configuration of the PLC, in the chapter [▶4.3.7.5 CANopen◀](#) from page 93 onward.

#### Expert tab

When Motion Control is used, the "Expert" tab is used to display the objects available in the CANopen slave and their values.

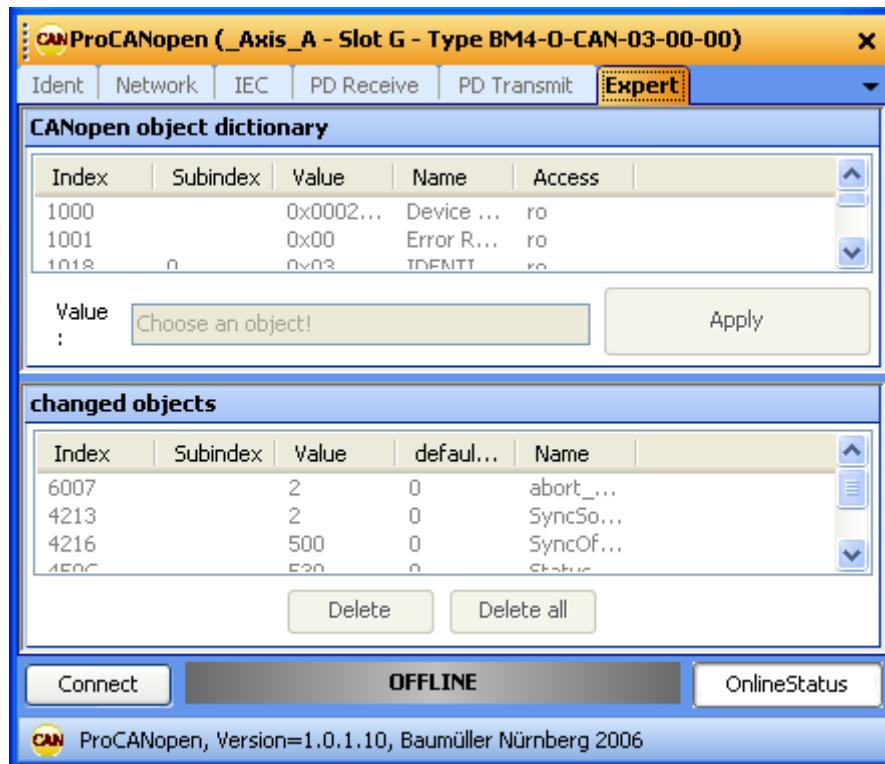


Figure 46: ProCANopen - CANopen slave communication configuration "Expert" tab

### 4.3.6.2 Configuring CANopen master communication

Open ProCANopen for our CANopen master by clicking on the device "BMC\_PLC\_1" and then select "Configuration Data (Components)\CANopen Master (Module number 1)\Configure Master Bus (ProCANopen)" via the context menu.

The CANopen master was already set with the default setting when creating the machine configuration so that the CANopen settings for Motion Control are configured.

The window for the CANopen master communication configuration is opened.

#### Ident tab

For Motion Control, you can set the module number of the CANopen master module on the b maXX controller PLC, the name of this module (device name) in ProMaster and the baud rate and the node-ID for CANopen in the "Ident" tab.

Changing the module number, device name, baud rate and node-ID affects the ProMaster project.

In addition, the technology and the CANopen-specific information like the device type, CANopen profile, manufacturer-ID etc. are also displayed.

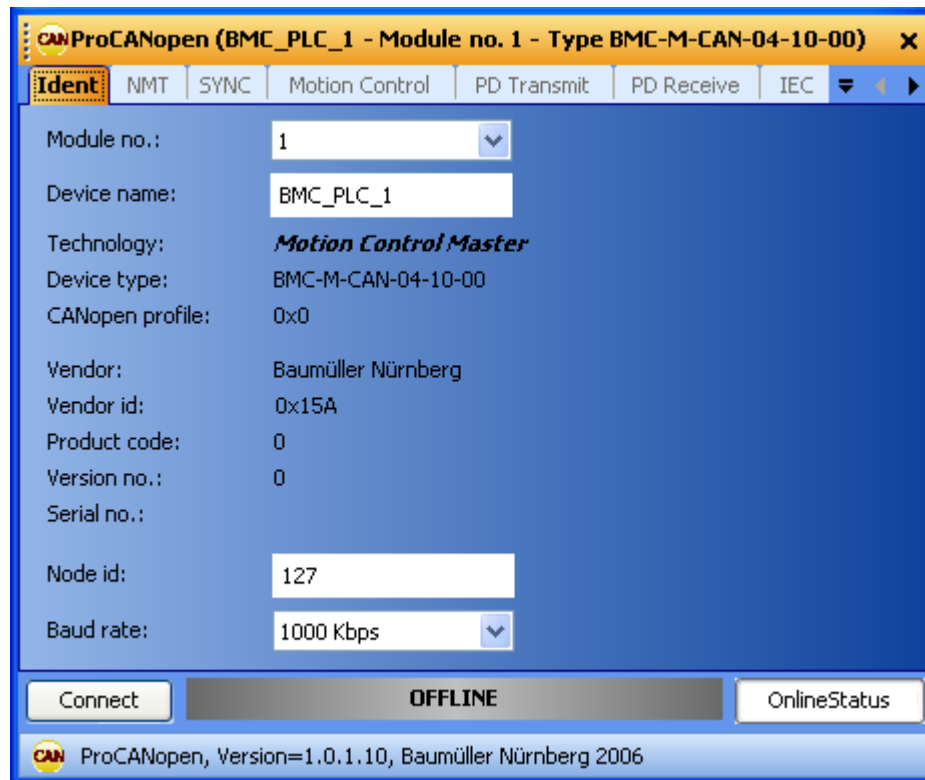


Figure 47: ProCANopen - CANopen master communication configuration Ident tab

### NMT tab

When using Motion Control, the settings on the "NMT" tab remain unchanged.

In the edit box "Waiting time of slaves", it can be set how long the CANopen master waits for the boot-up telegram of the CANopen slaves following switch-on. The entry 0 ms means that the master waits until all slaves have transmitted their boot-up message. This means the master will not signal a timeout.



### NOTE

There are CANopen slaves that do not transmit their boot-up telegram until over 30 seconds after being switched on.

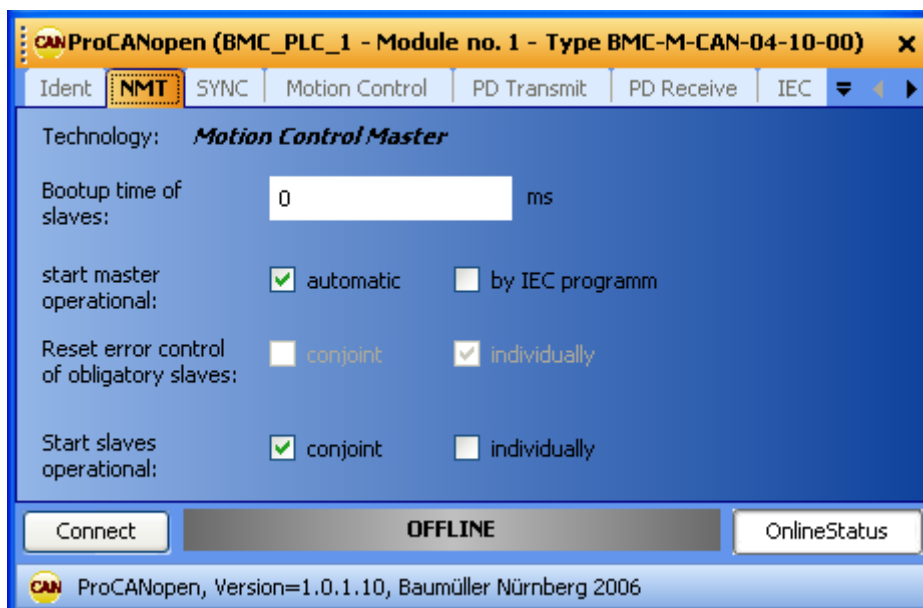


Figure 48: ProCANopen - CANopen master communication configuration NMT tab

### SYNC tab

When using Motion Control, you can set the SYNC cycle time and the synchronization signal between the CANopen master module and the b maXX controller PLC in the "SYNC" tab.

The change from the SYNC cycle time and the synchronization signal affects the ProMaster project.

It must be noted that the CANopen master transmits the SYNC telegram on the CANopen bus as a SYNC generator. The transmission interval is specified for the CANopen master in our example via the signal in the "Synchronization with PLC (Event)" combination box. For the CANopen master module you can choose from the signals "SYNC-Signal 1" and "SYNC-Signal 2". Other signals are not permitted. The signal must be generated on the b maXX controller PLC. With Motion Control, the signal "SYNC-Signal 1" is used and automatically generated by the b maXX controller PLC in the default configuration.

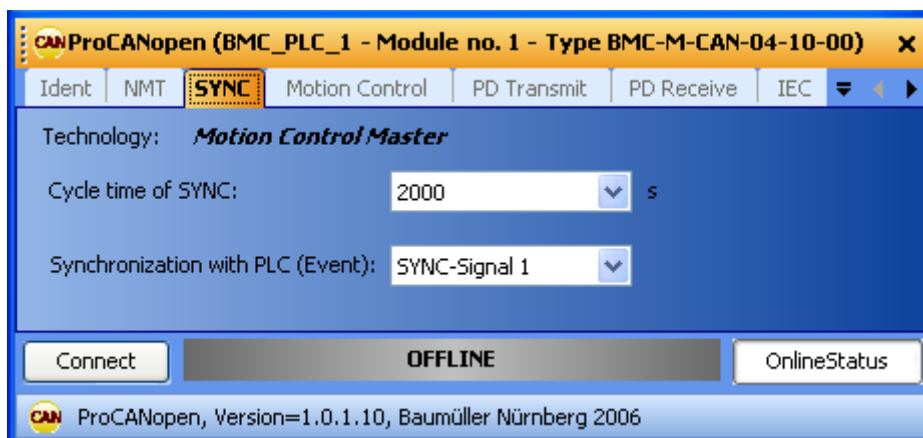


Figure 49: ProCANopen - CANopen master communication configuration SYNC tab

### Motion Control tab

The settings for Motion Control are made in the "Motion Control" tab.

For the machine configuration, you can specify whether Motion Control will be used. The selection is enabled for you when the master is selected from the project tree.

In addition, you can specify for individual drives that these are not controlled via Motion Control and, for example, change the names of individual drives. The selection is enabled for you when the respective drive is selected in the project tree. The changes affect the ProMaster project.

### NOTE



If you use drives without Motion Control in the machine configuration, these must always be configured (see [>Configuring CANopen slave communication<](#) in Chapter "4.3 ProMaster, ProCANopen, ProProg wt III and Motion Control" from Page 66.

In addition, you are shown Motion Control-specific information, such as the Motion Control cycle time and the CANopen-specific information, such as the node-ID of the respective drive.

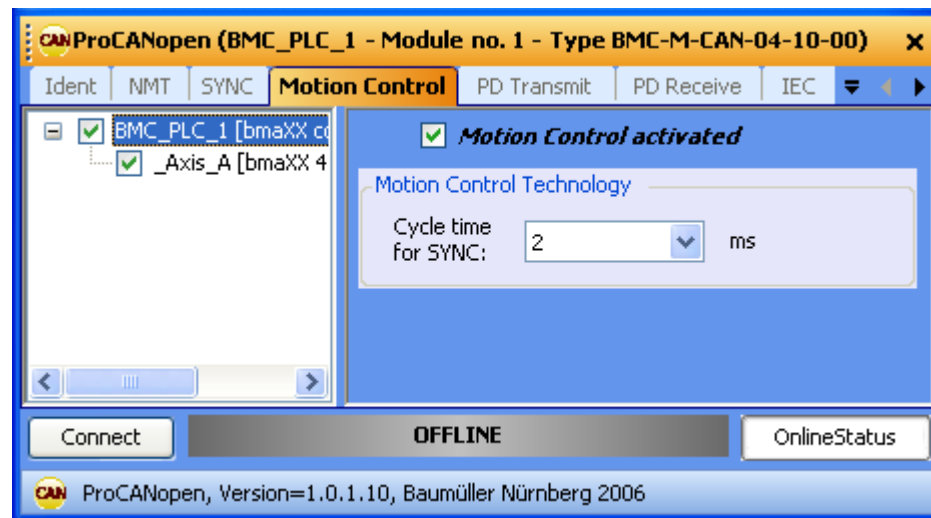


Figure 50: ProCANopen - CANopen master communication configuration Motion Control tab

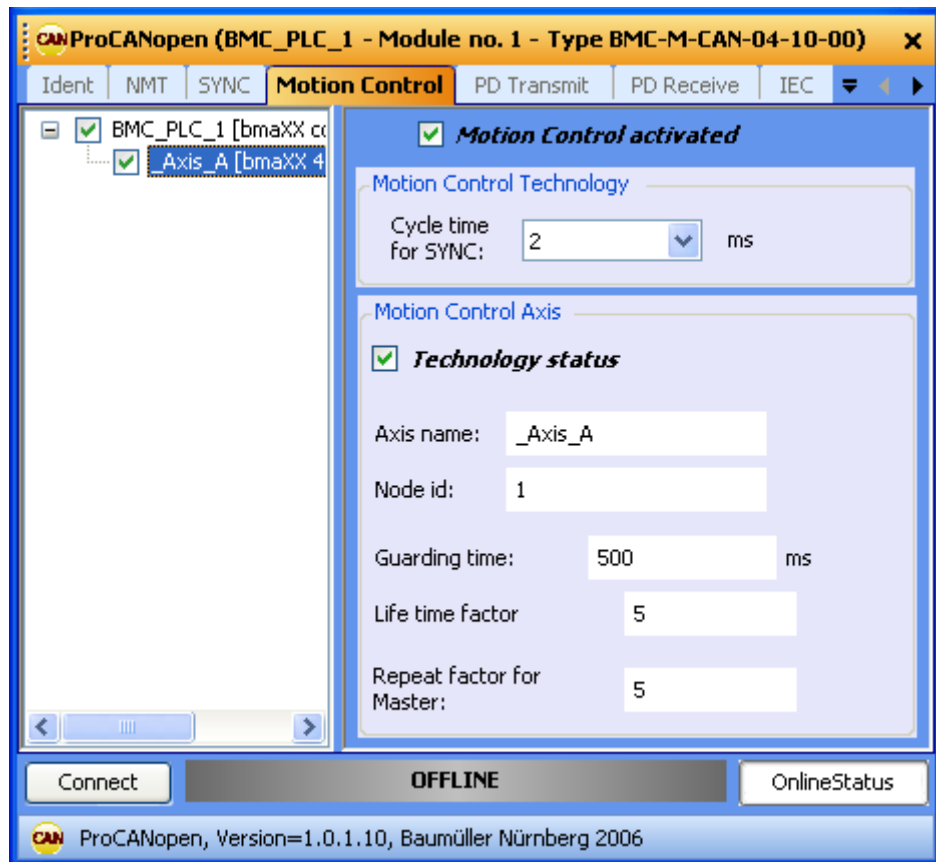


Figure 51: ProCANopen - CANopen master communication configuration Motion Control tab

### PD Transmit and PD Receive tab

The data on these pages are used for the display.

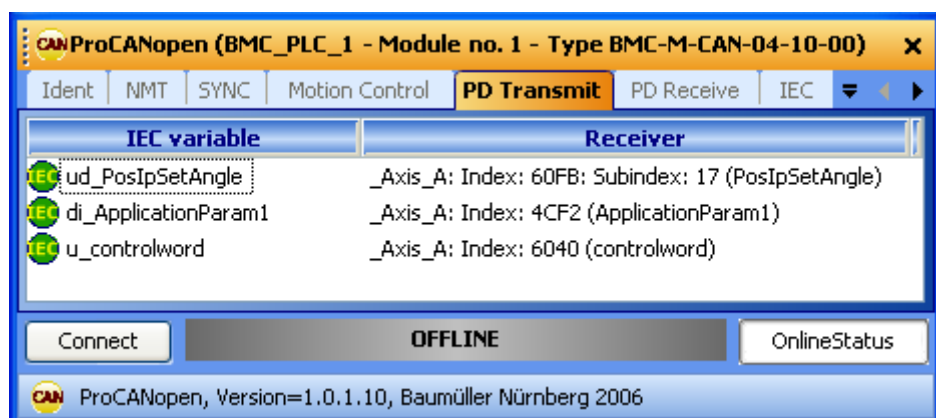


Figure 52: ProCANopen - CANopen master communication configuration PD Transmit tab

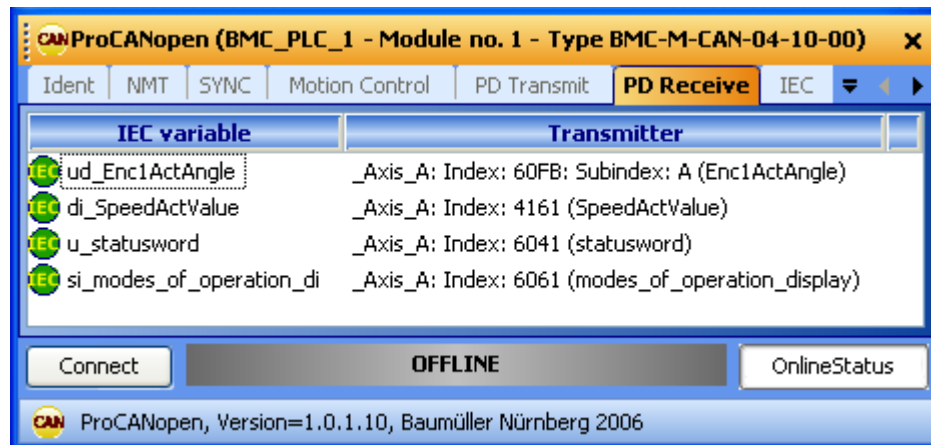


Figure 53: ProCANopen - CANopen master communication configuration PD Receive tab

In the "PD Transmit" tab and "PD Receive" tab the IEC variables (network variables), are each shown for transmission (PD Transmit) and reception (PD Receive) respectively.

The network variables (via the CANopen master objects) are assigned to the CANopen slave objects automatically with ProMaster and ProCANopen.

Additional CANopen slave objects are assigned to network variables during the configuration of the CANopen slave communication of the CANopen slaves. After this configuration of the CANopen slaves, the corresponding data are shown in the „PD Transmit" tab and the „PD Receive" tab of the configuration of the CANopen master communication.

To configure the CANopen communication of the CANopen slaves, see the chapter [4.3.6.1 Configuring CANopen slave communication](#) from page 66 onward.

### IEC tab

During the configuration of the CANopen slave communication, the CANopen slave objects (via CANopen master objects) are assigned to the network variables for the IEC project on the PLC.

The variables shown in the IEC tab are also called network variables. The process data are written and read in the IEC project via the network variables.

When using Motion Control, some of the assignments are created with the default settings.

The network variables and their data type in the IEC project on the PLC are shown in the "IEC" register.

The calculation of the (IEC) address of the network variables is not carried out until after "Update list" in the chapter [Download tab](#) in Chapter "4.3.6.2 Configuring CANopen master communication" from page 80 onward.

To configure the CANopen communication of the CANopen slaves, see [4.3.6.1 Configuring CANopen slave communication](#) from page 66 onward.

## 4.3 ProMaster, ProCANopen, ProProg wt III and Motion Control

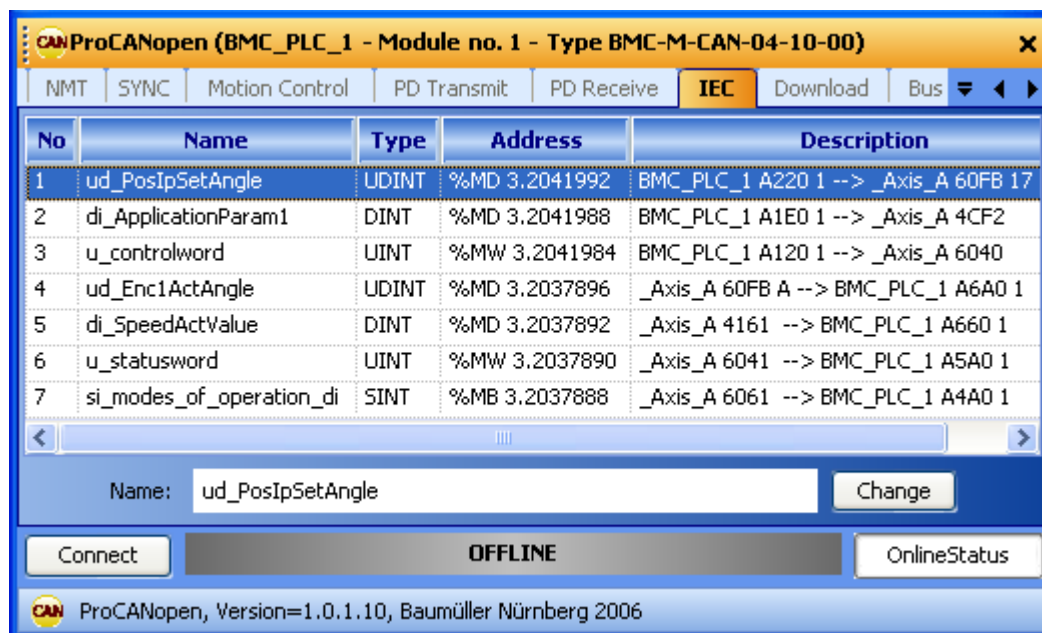


Figure 54: ProCANopen - CANopen master communication configuration IEC tab following configuration of the CANopen slaves and "Update list" (Download tab)

You can change the name of the network variables by selecting the network variable name from the "Name" column, editing the new network name in the "Designation" edit box and then clicking on the "Change" button.

### NOTE



Please take the limit of 30 characters for a variable name in the IEC project in ProProg wt III into account.

### Download tab

In the Download tab, the objects are shown which are loaded on the CANopen master (Download) following the configuration of the CANopen communication of the CANopen master and of the CANopen slave. As long as the configuration is not completed, no data are shown in the Download tab, i.e. no data can be loaded on the CANopen master.

Now press the "Update list" button (lower right). This generates the configuration data (both master and slave) for the download on the CANopen master.



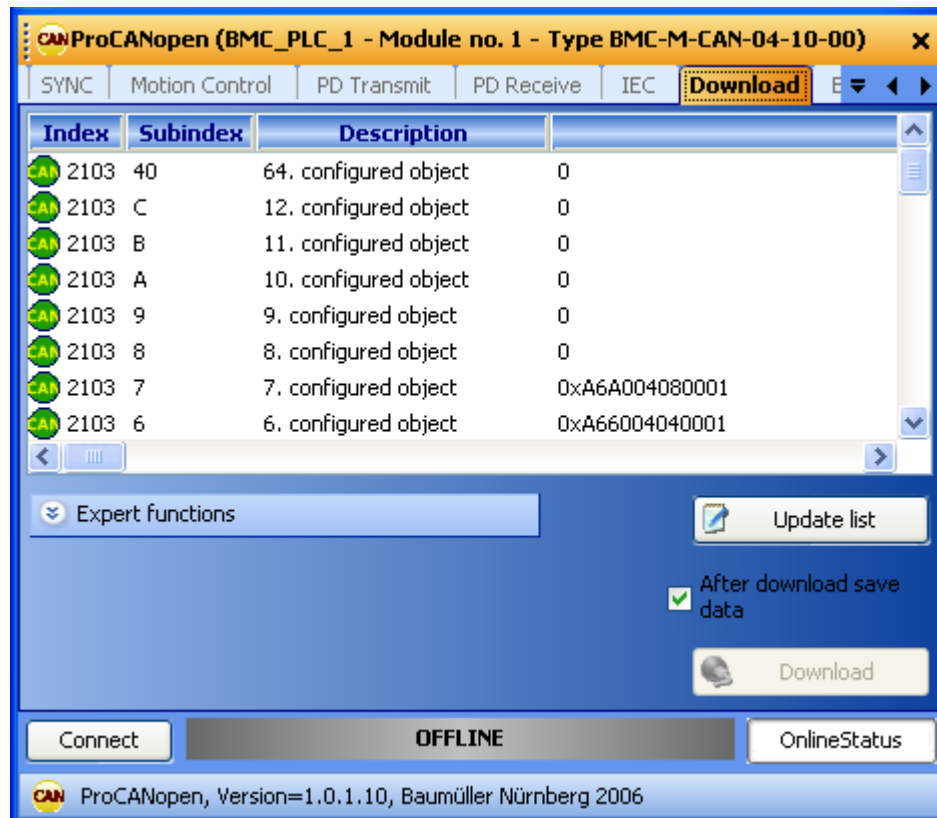


Figure 55: ProCANopen - CANopen master communication configuration Download tab

Now press the "Download" button. The connection from ProMaster to the CANopen master (BMC-M-CAN04 module) is established via the b maXX controller PLC (BMC-M-PLC01 module). Then the CANopen configuration data (both master and slave) are transmitted on the CANopen master.

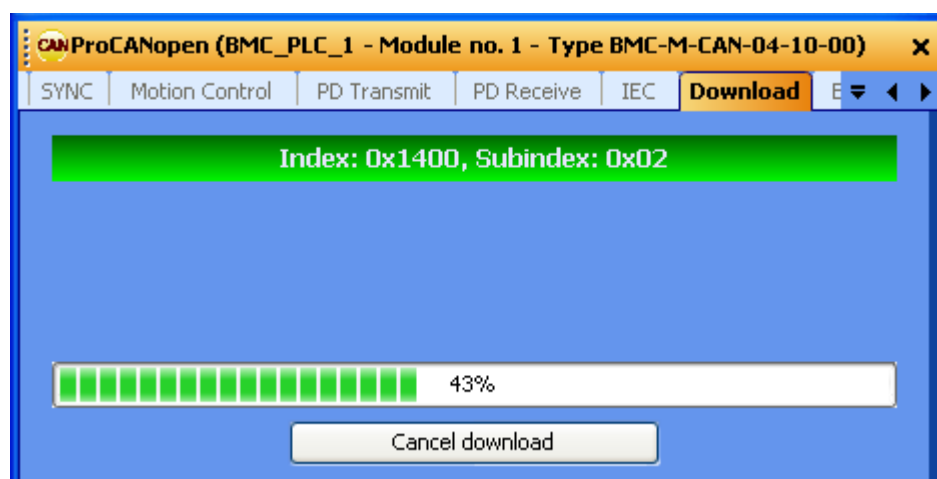


Figure 56: ProCANopen - CANopen communication configuration Download

If errors are reported here, it may be necessary to switch the CANopen master into the CANopen "Reset" state with the "Bus Control" tab, "Establish connection" button and "Reset" button, and then to start the download again in the "Download" tab.

After the download is completed, the Download object list is shown again.

Now both the CANopen configuration data for the CANopen master and the CANopen configuration data for the CANopen slave, which the master transmits to the slave when the CANopen network is started up, are located on the CANopen master.

### Bus control tab

In the "Bus control" tab, experienced CANopen users can control the CANopen network "by hand" following the download of the CANopen configuration data. This can, for example, be helpful during the physical commissioning of the CANopen network.

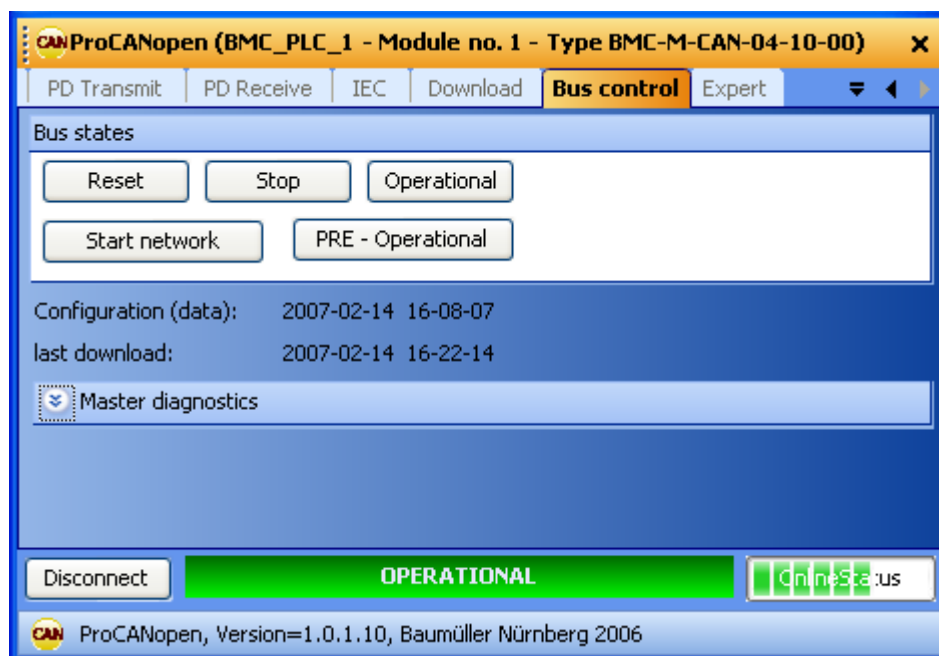


Figure 57: ProCANopen - CANopen master communication configuration "Bus control" tab

### 4.3.7 Configuring PLC

After the configuration of the CANopen network has been configured and the IEC project has been created, the data for the b maXX controller PLC are now configured.

In the process,

- the IEC project is integrated in the ProMaster project
- the cam data records are selected.

In addition, it is possible to

- create your own cam disk data with the ProCAM cam disk editor
- carry out the IO configuration with the ProDevice IO configurator

Then you click on the "Update entire IEC project" button in the "ProPLC" window. The data for the machine configuration, both for the CANopen network and for the b maXX controller PLC, are then generated.

#### 4.3.7.1 IEC

In this section we explain how an IEC project is connected to ProMaster.

Open the window "PLC Configuration" in the ProMaster project in the network view for our CANopen master by clicking on the device "BMC\_PLC\_1", selecting "Configuration Data (Components)\PLC (Module number 0)\PLC - Configuration (ProPLC)" via the context menu and then selecting the "IEC" tab there.

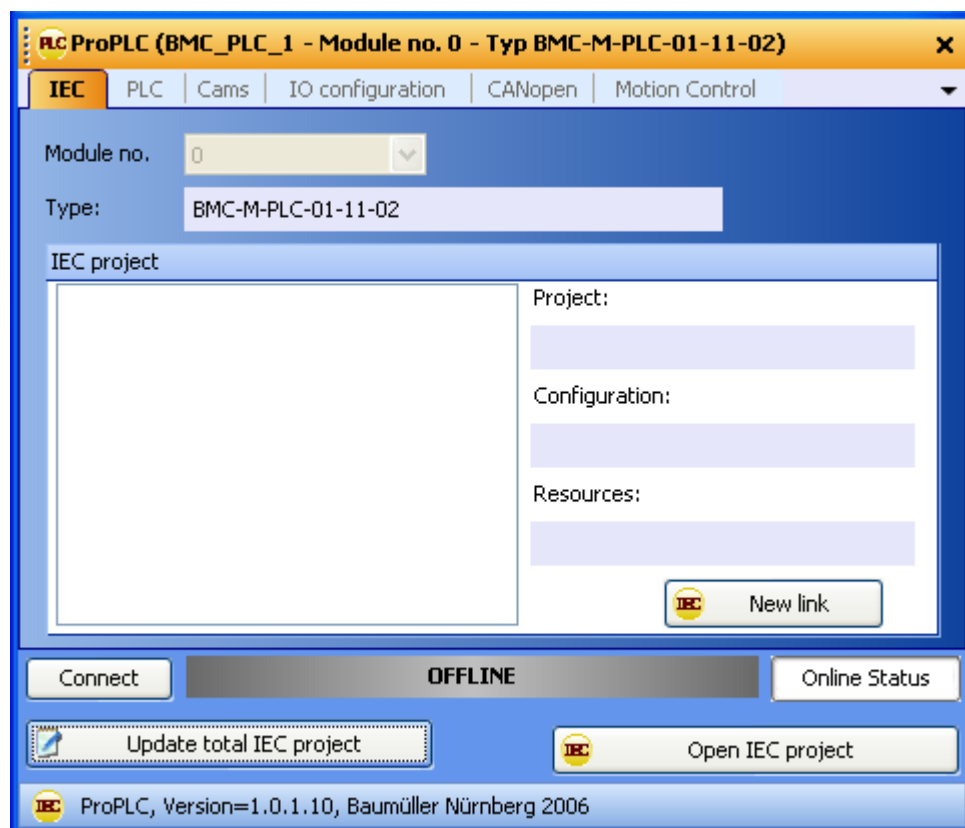


Figure 58: ProPLC - Connecting to example IEC project



#### NOTE

Only ProProg wt III projects can be opened with ProMaster. If you want to use an existing PROPROG wt II project, you must first open it with ProProg wt III (and convert it in the process) and can then open and use it in ProMaster.

Please note that your PROPROG wt II libraries are also converted in the process!

## 4.3 ProMaster, ProCANopen, ProProg wt III and Motion Control

Click on the "New link" button and select our example IEC project "Example\_BMC\_M\_CAN04\_MA\_1.mwt", which we created in the chapter [▶4.3.4 Creating a IEC project with ProProg wt III](#) from page 59 onward.

The IEC project just selected is shown in the "IEC" tab.

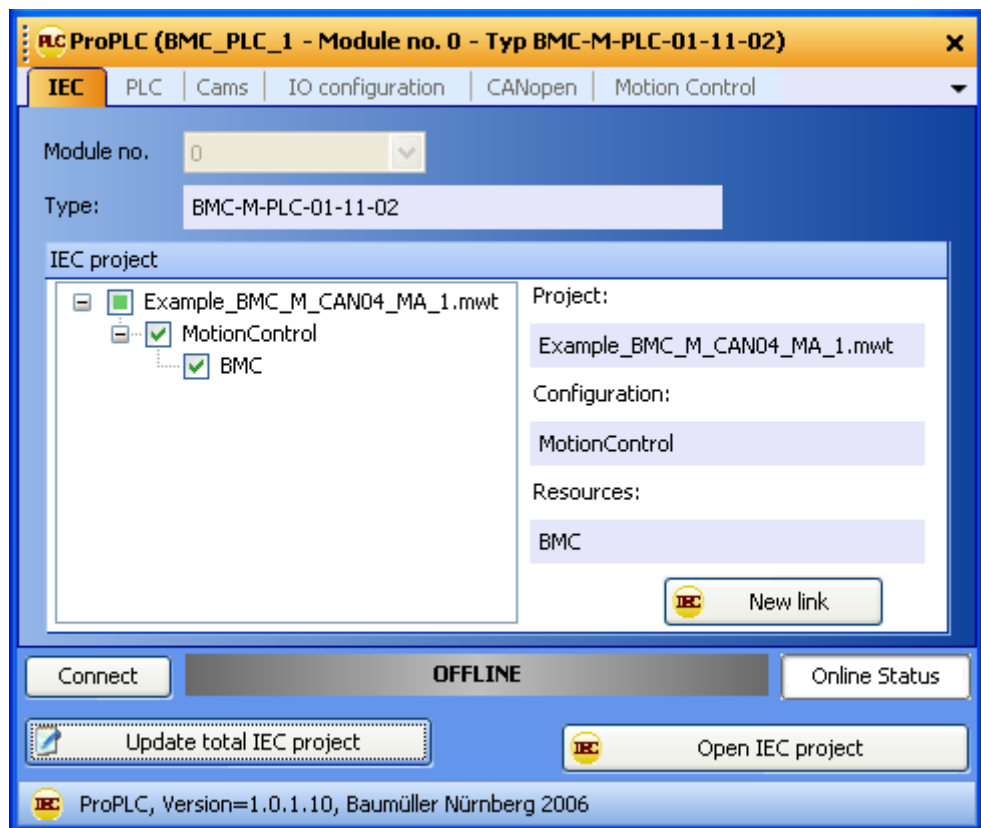


Figure 59: ProPLC - IEC tab

If you connect an existing application (or a template) from ProProg wt III to the ProMaster project, the names of the devices in ProMaster and in the IEC project may not be identical. In this case, see the chapter [▶4.3.9 Programming IEC project](#) from page 96 onward.

### 4.3.7.2 PLC

Following downloading data to CANopen master (see [▶Downloading data to CANopen master and to b maXX controller PLC](#) from page 95), the IEC project in the PLC tab can be activated and started on the b maXX controller PLC (area „PLC status“).

Furthermore data of the b maXX controller PLC, such as version of operating system, Firmware version, IEC project and boot project are shown in the area „PLC Info“. Information to the files in the Flash memory of the b maXX controller PLC are shown in the area „Flash directory“ (following „Connect“ and „Update“), e.g. cam data set (see [▶4.3.7.3 Cam disks](#)).

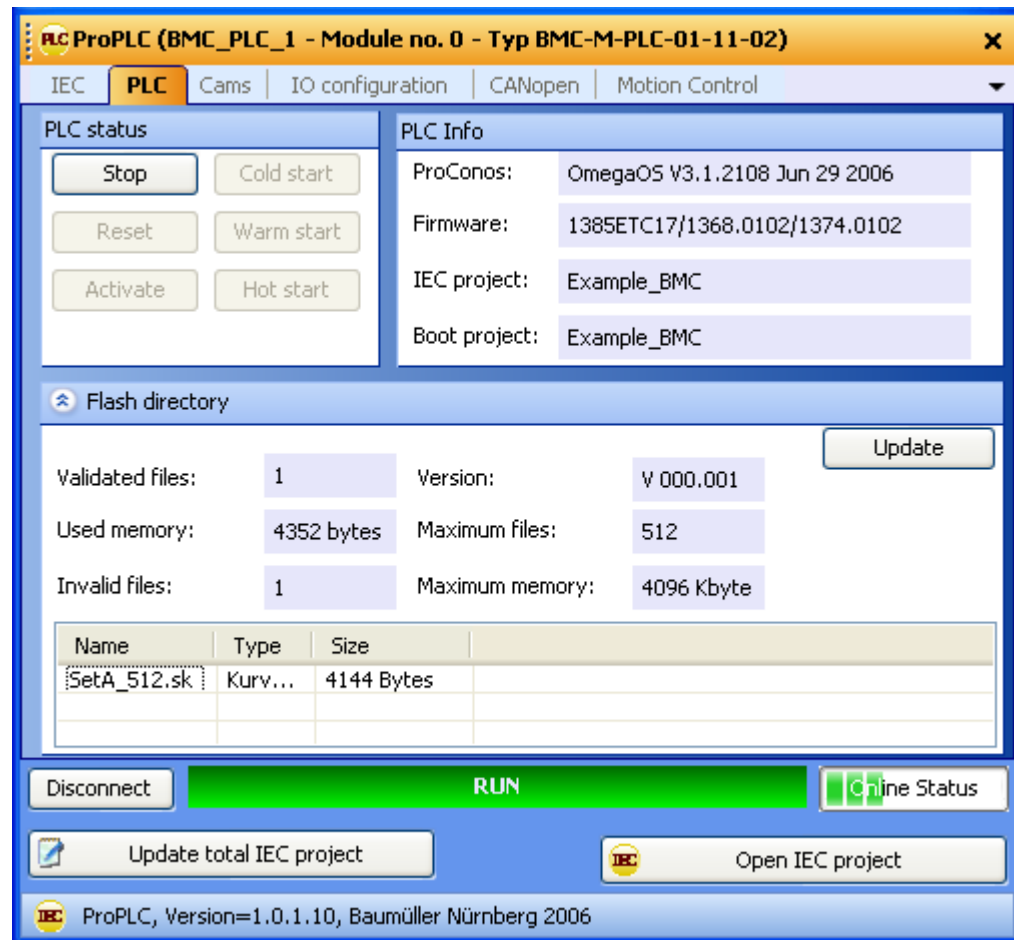


Figure 60: ProPLC - PLC tab following download of cam disk (cam disk tab)

#### 4.3.7.3 Cam disks

In this section we explain how a cam-disk data record is created and connected to ProMaster.

Open the window "PLC Configuration" in the ProMaster project in the network view for our CANopen master by clicking on the device "BMC\_PLC\_1", selecting "Configuration Data (Components)\PLC (Module number 0)\PLC - Configuration (ProPLC)" via the context menu and then selecting the "Cams" tab there.

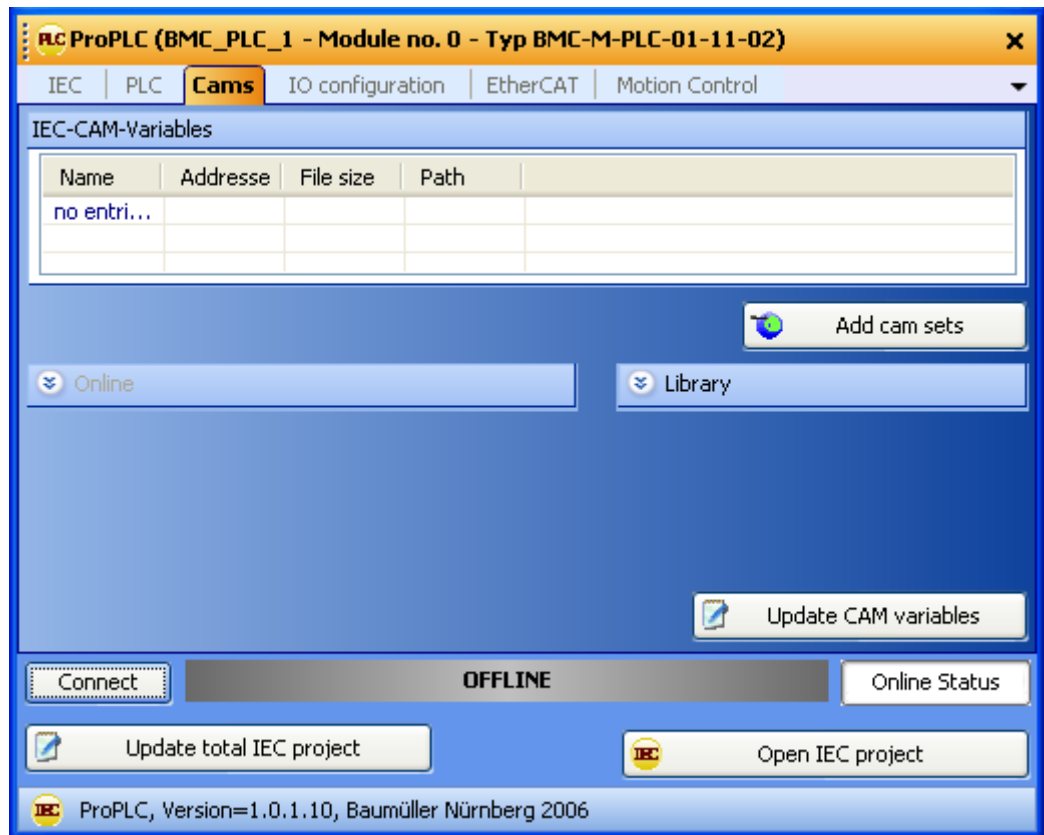


Figure 61: ProPLC - "Cam Disks" tab without cam data record

Press the "Add cam sets" button. The "ProMaster.NET - Cam Disks" window is opened. Press the "Add" button in the "Hard disk" area and select the path on your hard disk in which your cam data are located.



### NOTE

Example cam data are contained in the installation directory of ProCAM in the sub-folder "examples" (e.g. C:\...\Baumueller\ProCam 2\examples).

If you want to generate your own cams, click on the "ProCAM" button to open the cam disk editor. You can edit your cams there.

Now put together your cam data record (in the area "\*.sk files") from the various cam disk data (in the area "Hard disk") with Drag&Drop.

For our example we select the cam disk data "ExampleGerade\_MC.kbin" and "ExamplePendelP5\_MC.kbin", which are grouped in the cam-disk data record "SetA\_512".

**NOTE**

You can graphically display the cam disk data (in the respective area) with the "Preview" button. This simplifies the cam selection.

Then click on the "Save" button. This creates the cam data record and makes it available to ProMaster.

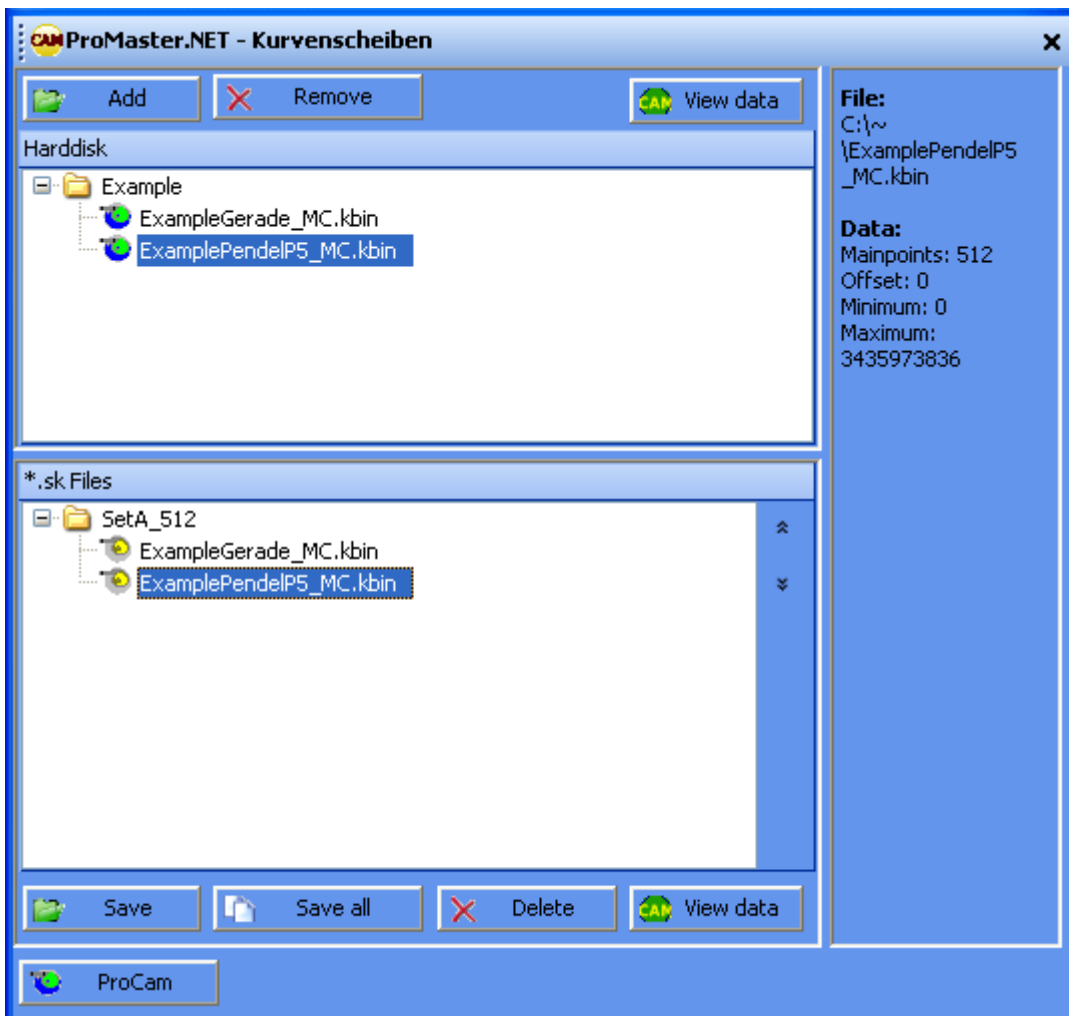


Figure 62: ProMaster (- PLC configuration - Cam Disks tab -) Create cam data record

Now close the "ProMaster.NET - Cam Disks" window by clicking on "x" at the upper right. In the "Cams" tab you now see the name of the variables of the cam-disk data record in the IEC project, their address in the IEC project and the file name of the cam-disk data record on the hard disk.

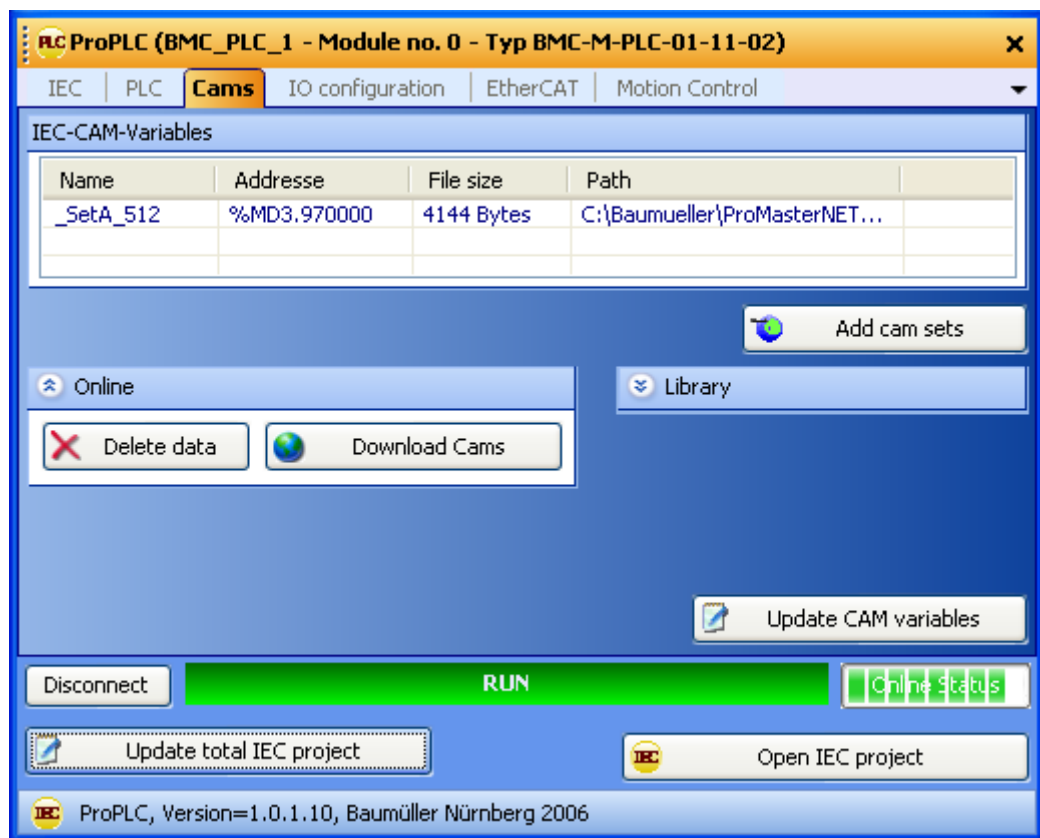


Figure 63: ProPLC - "Cam Disks" tab with cam data record

If you connect an existing application (or a template) from ProProg wt III to the ProMaster project, the cam data-record name in ProMaster and in the IEC project may not be identical. In this case, see the chapter [4.3.9 Programming IEC project](#) from page 96 onward.

From within the "Cams" tab you can clear the flash memory on the b maXX controller PLC with the "Delete data" button. This may be necessary if the flash memory is "full" due to various downloads (of both cam-disk data records and IEC projects).

In addition, you can download the cam-disk data records individually onto the b maXX controller PLC from within the "Cams" tab via the "Download Cams" button.

Now load the cam-disk data records onto the b maXX controller PLC via the "Download Cams" button.

### 4.3.7.4 IO configuration

In this section we explain how an IO configuration for the b maXX controller PLC is created and connected to ProMaster.

If you want to operate a b maXX controller PLC with the CANopen master module without IO modules, you need not carry out any configuration in the "PLC - Configuration" window, "IO Configuration" tab.



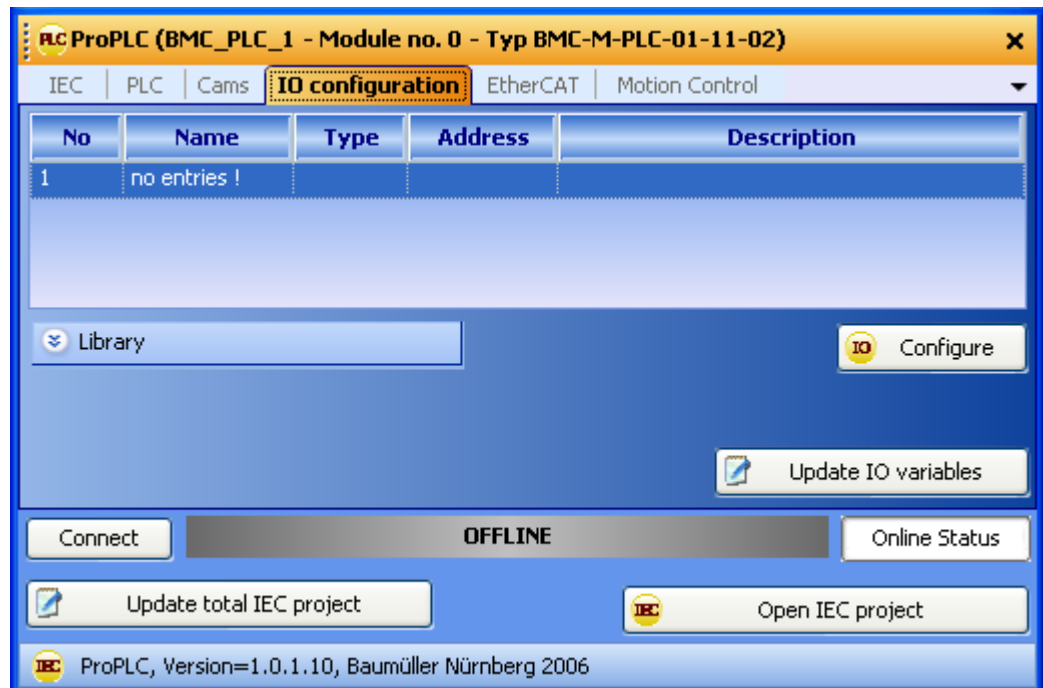


Figure 64: ProPLC - "IO Configuration" tab

If you want to operate a b maXX controller PLC with a CANopen master module and IO modules, open the IO module configurator by clicking on the device "BMC\_PLC\_1" in the ProMaster project in the network view, and then selecting the "Configuration Data (Components)\PLC (Module number 0)\IO - Configuration (ProDevice)" via the context menu. In ProMaster this opens the "Device view" of the device "BMC\_PLC\_1".

## 4.3 ProMaster, ProCANopen, ProProg wt III and Motion Control

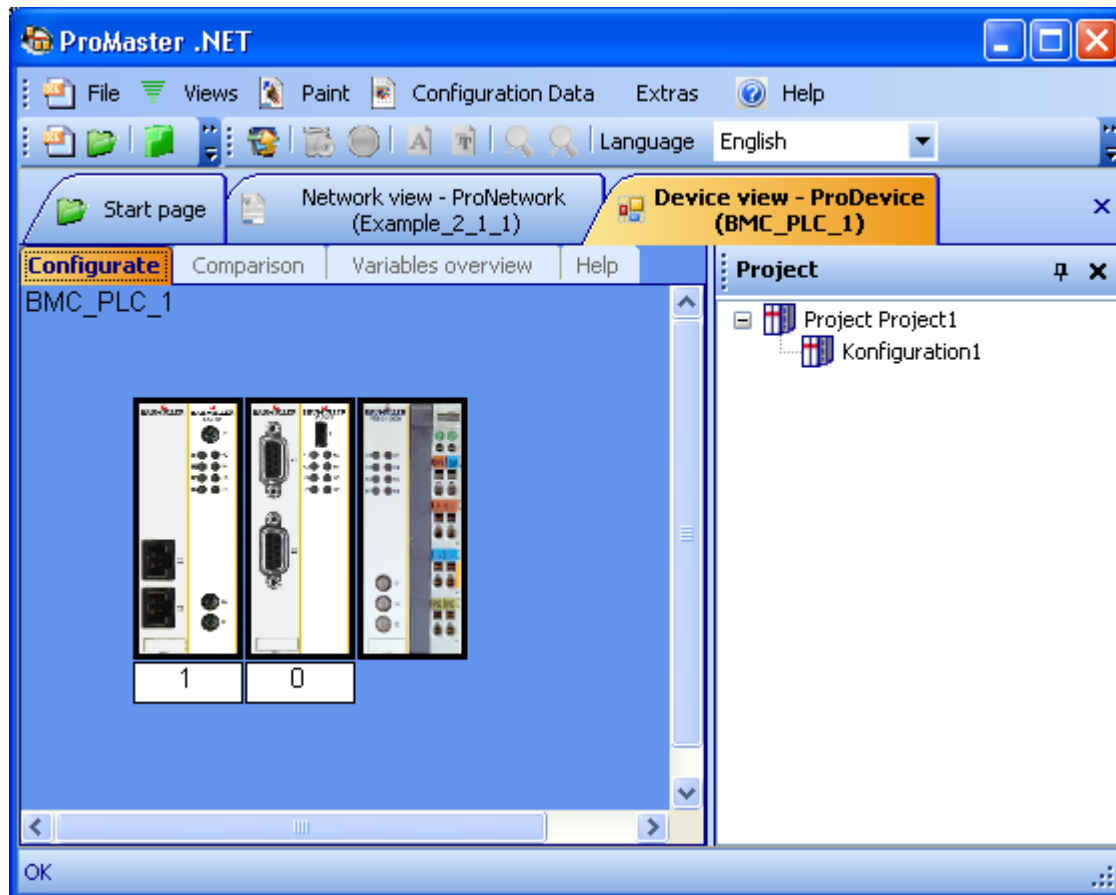


Figure 65: ProMaster - IO Configuration 1

Now open the Baumüller catalog via the "Views\Catalog" menu. Select the "IO Modules" tab.

We use the following IO modules for our example:

- DO8000 (8 digital outputs; from the "digital" group)
- DI8000 (8 digital inputs; from the "digital" group)

Now position the IO module DO8000 (to the right of "BMC\_PLC\_1") with Drag&Drop. The end terminal is automatically positioned to the right of the IO module. Then drag the IO module DI8000 between "DO8000" and the end terminal (now to the right of "DO8000").

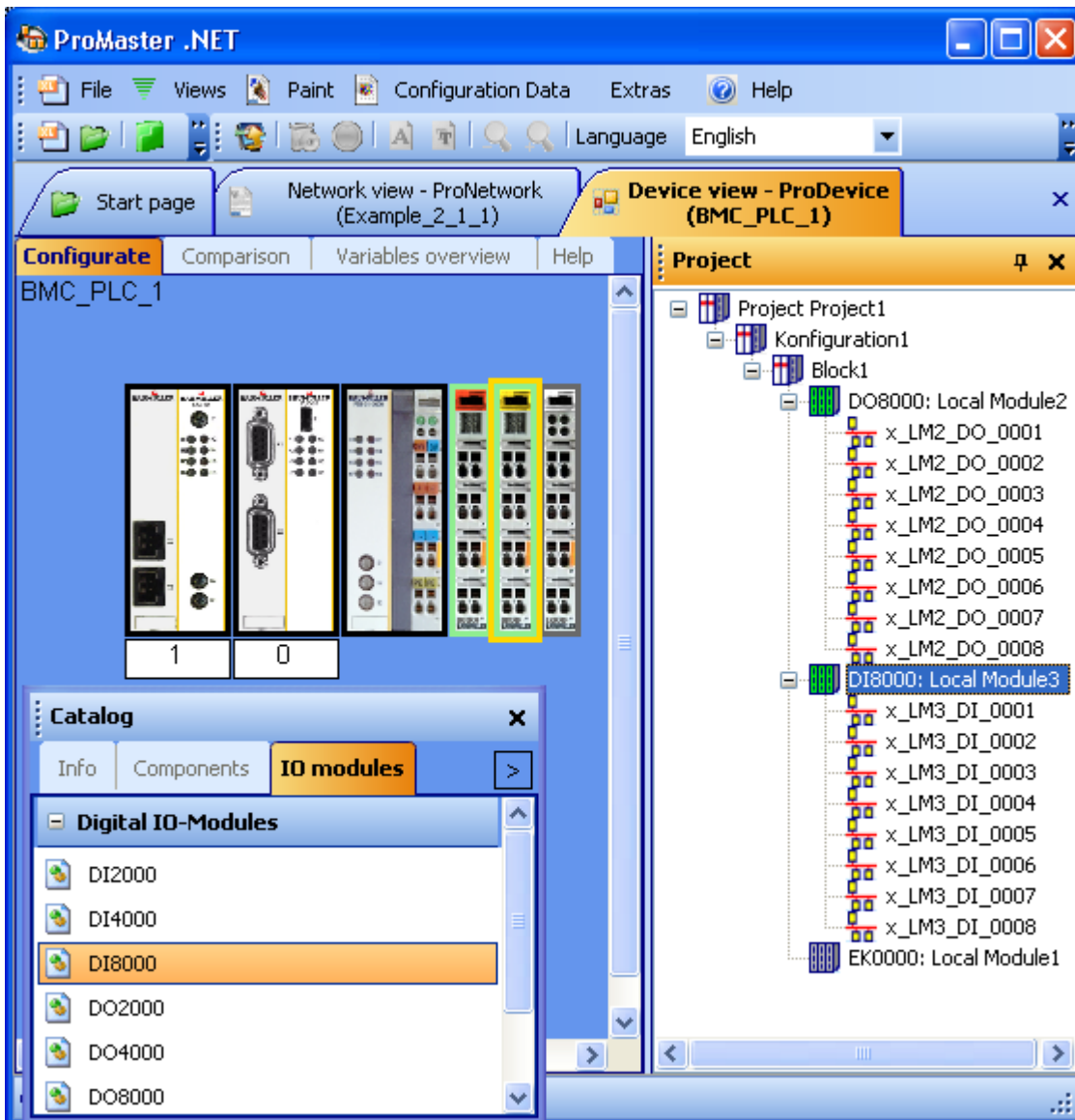


Figure 66: ProMaster - IO Configuration 2

In "Configure" you see the IO modules and their individual inputs or outputs. The IEC variable names of the individual inputs and outputs can be changed both in "Configure" and in the "IEC - Variables" tab of the "Device view".

The "Catalog" window is closed for improved clarity.

Now select the "Comparison" tab in the "Device view" and apply the IO configuration just put together in the IEC project for ProProg wt III. Use the arrows for this purpose.

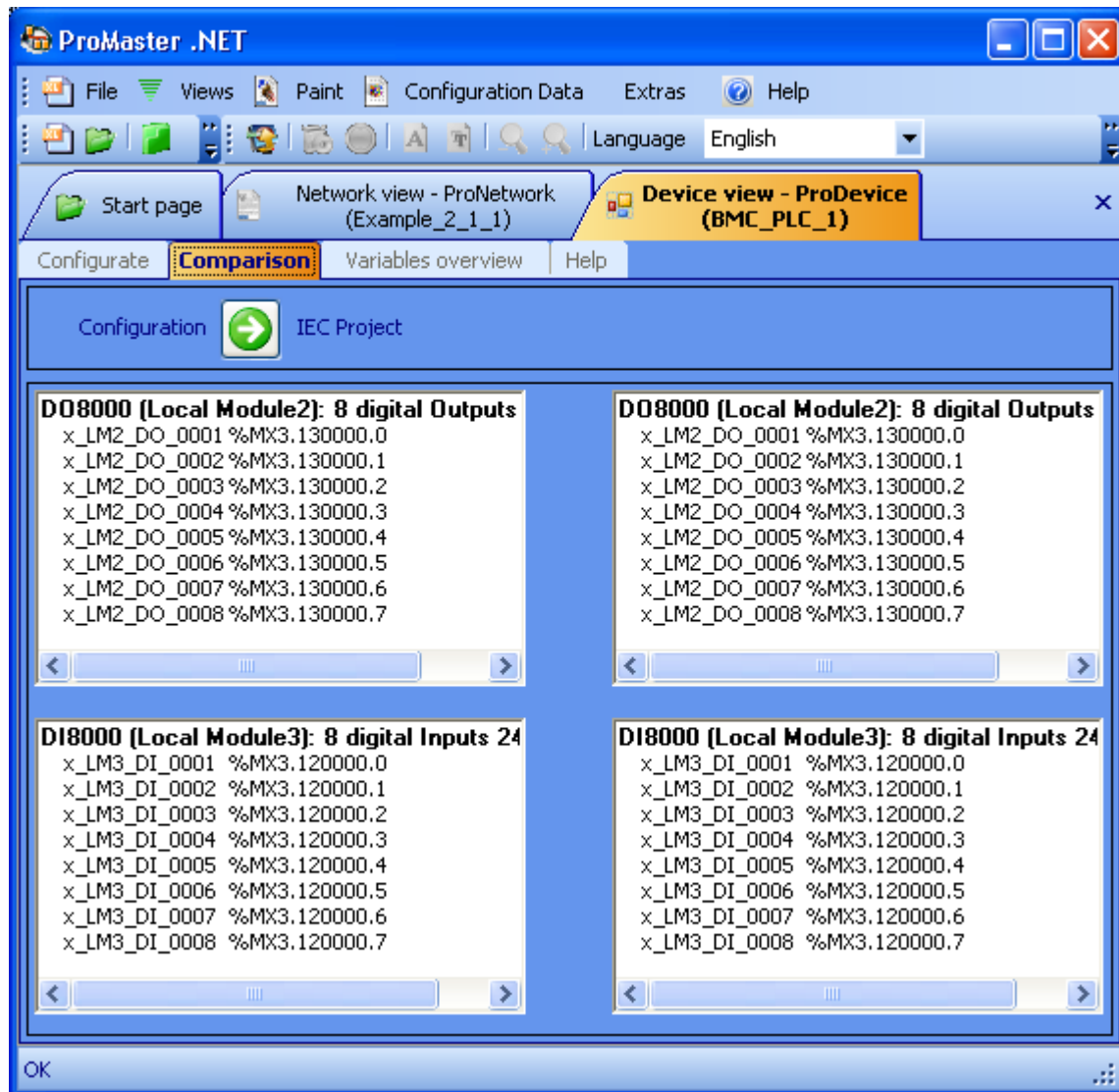


Figure 67: ProMaster - IO configuration comparison

Now switch back to the "PLC Configuration" window again. If you have closed it, open it by clicking on the device "BMC\_PLC\_1" and selecting "Configuration Data (Components)\PLC (Module number 0)\PLC - Configuration (ProPLC)" via the context menu.

In the "IO Configuration" tab the variables in the IEC project are shown with a different name, address and comments.

The abbreviation "\_LM\_" in the default IO variable name stands for "Local Module".

The IEC variables are written to the IEC project (global variable worksheet "Global\_Variables") when the "Update IO variables" button is pressed in "PLC - Configuration" in the "IO Configuration" tab.

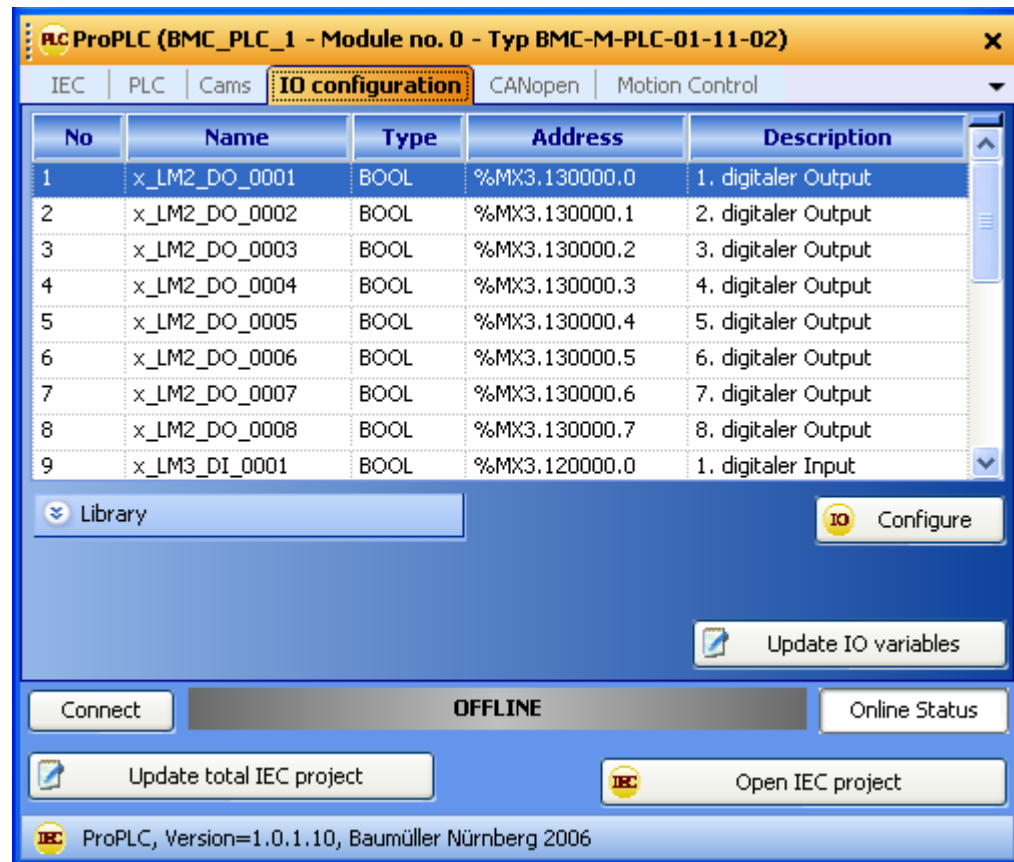


Figure 68: ProPLC - IO with DI and DO tab

#### 4.3.7.5 CANopen

In the "CANopen" tab the name, address and comments of the network variables of the CANopen slaves in the IEC project are shown.

When using Motion Control, the communication is carried out via the axis variable. After the "Update entire IEC project" button is pressed, see the chapter [▶4.3.8 Downloading data to CANopen master and to b maXX controller PLC](#) from page 95 onward, ProMaster creates the new axis variable with the device name of the CANopen slave in the IEC project. In addition, ProMaster creates network variables in the IEC project. These are divided into the standard network variables for Motion Control and, if additional objects have been created during the CANopen slave configuration, the additional network variables for CANopen.

These network variables are shown in the "CANopen" tab, i.e. both the standard network variables for Motion Control and the additional network variables for CANopen.

The following must always be observed here:

The standard network variables for Motion Control may be read by the user, but not written.

These are the standard Motion Control network variables for the setpoint values:

"u_controlword.."	for the axis control word
"ud_PoslpSetAngel.."	for the synchronized position angle setpoint value of the axis

## 4.3 ProMaster, ProCANopen, ProProg wt III and Motion Control

These are the standard Motion Control network variables for the actual values:

"u_statusword.."	for the axis status word
"ud_Enc1ActAngle.."	for the synchronized position angle actual value of the axis
"si_modes_of_operation_display.."	for the axis operating mode

### NOTE



The standard network variables for Motion Control may be read by the user, but not written.

### NOTE



The network variables must be read and written in the Motion Control event task.

Exception: The PDO transmission type of the network variables is set asynchronous.

The number after the network variable name is an internal number to distinguish automatically generated network variables with the same name.

No	Name	Type	Address	Description
1	ud_PosIpSetAngle	UDINT	%MD 3.2041992	BMC_PLC_1 A220 1 --> _Axis_A 60FB 17
2	di_ApplicationParam1	DINT	%MD 3.2041988	BMC_PLC_1 A1E0 1 --> _Axis_A 4CF2
3	u_controlword	UINT	%MW 3.2041984	BMC_PLC_1 A120 1 --> _Axis_A 6040
4	ud_Enc1ActAngle	UDINT	%MD 3.2037896	_Axis_A 60FB A --> BMC_PLC_1 A6A0 1
5	di_SpeedActValue	DINT	%MD 3.2037892	_Axis_A 4161 --> BMC_PLC_1 A660 1
6	u_statusword	UINT	%MW 3.2037890	_Axis_A 6041 --> BMC_PLC_1 A5A0 1
7	si_modes_of_operation_di	SINT	%MB 3.2037888	_Axis_A 6061 --> BMC_PLC_1 A4A0 1

Figure 69: ProPLC - CANopen tab

If during the configuration of the CANopen slave communication you have created the additional network variables (and their link to CANopen slaves) suggested in the sections [▶PD Receive tab◀](#) from page 69 and [▶PD Transmit tab◀](#) from page 71, you see the additional network variables.

The additional network variable for the setpoint values is:

"di\_ApplicationParam1.." for Application parameter 1 of the axis

The additional network variable for the actual values is:

"di\_SpeedActValue.." for the actual speed value of the axis

When changing network variable names, please note that each variable name can only be assigned once in the IEC project. This must especially be observed when using several CANopen master modules on the b maXX controller PLC.

---

**NOTE**

Each variable name may only be used once in the IEC project.

---

#### 4.3.7.6 Motion Control

---

In the future, the general settings for Motion Control will be configured in the "Motion Control" tab.

If you want to use the default Motion Control settings, you need make no settings here.

#### 4.3.8 Downloading data to CANopen master and to b maXX controller PLC

---

The data for the CANopen master is currently downloaded via [▶Download tab◀](#) in Chapter "4.3.6.2 Configuring CANopen master communication" from page 80 onward.

The cam data records for the b maXX controller PLC are currently downloaded via the "Cam Disk" tab of the PLC configuration. See the chapter [▶4.3.7.3 Cam disks◀](#) from page 85 onward.

The IEC project for the b maXX controller PLC is currently downloaded as usual via the ProProg wt III.

Now click on the "Update entire IEC project" button in the "PLC Configuration" window. The data for the b maXX controller PLC are now generated. This process can take some time, as among other things ProProg wt III is opened and the configured IEC variables are written to the global variable worksheet in the IEC project.

Afterwards you can compile the IEC project by confirming the respective request with „Yes“.

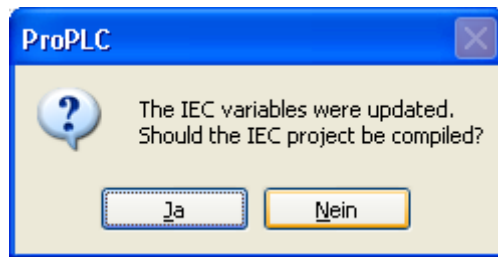


Figure 70: ProMaster - Compiling the updated IEC project

You can alternatively compile the IEC project also in ProProg wt III via the ProProg wt III menu "Code\Create New Project".

Now download the IEC project to the b maXX controller PLC with (ProProg wt III - "On-line\Project Check..." menu → "Transmit" → "Transmit" boot project).

Now carry out a reset on the b maXX controller PLC and then switch the PLC into the "RUN" state (as an alternative, you can also switch the entire CANopen network off and then on again). Now you can control the CANopen slave axis `_Axis_A` via the IEC project in ProProg wt III.

### 4.3.9 Programming IEC project

---

#### 4.3.9.1 General information

---

For information on how to program a Motion Control application in the IEC project in ProProg wt III, please see the Motion Control application manual and the ProProg wt III application manual.

ProProg wt III with our IEC project "Example\_BMC\_M\_CAN04\_MA\_1.mwt" is opened via the context menu on BMC\_PLC\_1 "Configuration Data\PLC\IEC - Programming". You can edit the IEC project in the accustomed manner.

ProMaster has written

- the Motion Control axis variable (in the section MC\_Axis\_Variables)
  - the cam-disk data record (in the section MC\_CamDataSet)
  - the network variables (in the section CANopenVariables)  
(standard network variables for Motion Control and additional network variables for CANopen; see [▶4.3.9.2 Data exchange◀](#) from page 98 onward)
  - the IO variables (in the section IOVariables)
- to the global variable worksheet "Global\_Variables".



Name	Type	Usage	Description	Address
<b>Global Variables</b>				
<b>MyApplication</b>				
_MyMaster	AXIS...	VAR_GLO...	Not from ProMaster	
<b>MC_CamDataSet</b>				
s_File_SetA_512	FileS...	VAR_GLO...		%MD3.970000
_SetA_512	MC_...	VAR_GLO...		%MD3.970000
<b>MC_AxisVariables</b>				
_Axis_A	AXIS...	VAR_GLO...	1. Achse	%MD3.451500
<b>MC_MasterRef</b>				
<b>MC_SystemReserved</b>				
<b>MC_AxisRef</b>				
<b>MC_BacInItRef</b>				
<b>MC_AxisPDORef</b>				
<b>MC_TriggerRef</b>				
<b>MC_IRPRef</b>				
<b>IOVariables</b>				
x_LM2_DO_0001	BOOL	VAR_GLO...	1. digitaler Output	%MX3.130000.0
x_LM2_DO_0002	BOOL	VAR_GLO...	2. digitaler Output	%MX3.130000.1
x_LM2_DO_0003	BOOL	VAR_GLO...	3. digitaler Output	%MX3.130000.2
x_LM2_DO_0004	BOOL	VAR_GLO...	4. digitaler Output	%MX3.130000.3
x_LM2_DO_0005	BOOL	VAR_GLO...	5. digitaler Output	%MX3.130000.4
x_LM2_DO_0006	BOOL	VAR_GLO...	6. digitaler Output	%MX3.130000.5
x_LM2_DO_0007	BOOL	VAR_GLO...	7. digitaler Output	%MX3.130000.6
x_LM2_DO_0008	BOOL	VAR_GLO...	8. digitaler Output	%MX3.130000.7
x_LM3_DI_0001	BOOL	VAR_GLO...	1. digitaler Input	%MX3.120000.0
x_LM3_DI_0002	BOOL	VAR_GLO...	2. digitaler Input	%MX3.120000.1
x_LM3_DI_0003	BOOL	VAR_GLO...	3. digitaler Input	%MX3.120000.2
x_LM3_DI_0004	BOOL	VAR_GLO...	4. digitaler Input	%MX3.120000.3
x_LM3_DI_0005	BOOL	VAR_GLO...	5. digitaler Input	%MX3.120000.4
x_LM3_DI_0006	BOOL	VAR_GLO...	6. digitaler Input	%MX3.120000.5
x_LM3_DI_0007	BOOL	VAR_GLO...	7. digitaler Input	%MX3.120000.6
x_LM3_DI_0008	BOOL	VAR_GLO...	8. digitaler Input	%MX3.120000.7
<b>CANopen Variables</b>				
ud_PosIpSetAngle	UDINT	VAR_GLO...	BMC_PLC_1 A220 1 --> _Axis_A 60...	%MD 3.2041992
di_ApplicationParam1	DINT	VAR_GLO...	BMC_PLC_1 A1E0 1 --> _Axis_A 4C...	%MD 3.2041988
u_controlword	UINT	VAR_GLO...	BMC_PLC_1 A120 1 --> _Axis_A 60...	%MW 3.2041984
ud_Enc1ActAngle	UDINT	VAR_GLO...	_Axis_A 60FB A --> BMC_PLC_1 A...	%MD 3.2037896
di_SpeedActValue	DINT	VAR_GLO...	_Axis_A 4161 --> BMC_PLC_1 A66...	%MD 3.2037892
u_statusword	UINT	VAR_GLO...	_Axis_A 6041 --> BMC_PLC_1 A5...	%MW 3.2037890
si_modes_of_operation_di	SINT	VAR_GLO...	_Axis_A 6061 --> BMC_PLC_1 A4...	%MB 3.2037888

Figure 71: ProProg wt III - Global variable worksheet with data from ProMaster

In particular, the following must be observed:

The device name of the device on the CANopen bus in ProMaster is simultaneously the name of the axes in the IEC project in ProProg wt III. That means that if you have connected an existing application (or a template) from ProProg wt III to the ProMaster

project, the names of the devices in ProMaster and in the IEC project may not be identical. In this case, there are two possible solutions:

- 1 ProMaster changes the axis names to the device names in ProMaster in ProProg wt III in the global variable worksheet "Global\_Variables" in the section "MC\_AxisVariables", and the user changes the axis names in the POEs to the device names in ProMaster (via the ProProg wt III "Global Replacement" function).
- 2 You change the device names in ProMaster to the axis names in ProProg wt III.

In our example we have given the CANopen slave the device name `_Axis_A` in ProMaster. This is also the axis name from our Motion Control template, which we have used for our IEC project.

The same also applies to the cam-data record name in the IEC project in ProProg wt III. That means that if you connect an existing application (or a template) from ProProg wt III to the ProMaster project, the cam data-record names in ProMaster and in the IEC project may not be identical. In this case, the following applies:

ProMaster changes the cam data-record names to the cam data-record names in ProMaster in ProProg wt III in the global variable worksheet "Global\_Variables" in the section "MC\_CamDataSet", and the user changes the cam data-record names in the POEs to the cam data-record names in ProMaster (via the ProProg wt III "Global Replacement" function).

### 4.3.9.2 Data exchange

---

A global variable is required for the data exchange between the function blocks in the IEC project on the b maXX controller PLC (BMC-M-PLC-01), the CANopen master module (BMC-M-ETH-02/BMC-M-CAN-04) and (via the CANopen field bus with) the CANopen slave. This variable is also called an axis variable and has an axis name in the IEC project. Our axis variable has the axis name `_Axis_A`. We have assigned our CANopen slave the (same) device name `_Axis_A` in ProMaster. Therefore no adaptation is required here.

The same applies to the variable for the cam data that has the cam data-record name `_SetA_512` in the IEC project and ProMaster.

The axis variable and the cam data variable are connected to the Motion Control function blocks in the IEC project. The machine function is programmed with the Motion Control function blocks.

Information on using the Motion Control function blocks is contained in the Motion Control application manual.

When using Motion Control, the communication is carried out via the axis variable. ProMaster creates network variables in the IEC project. These are divided into the standard network variables for Motion Control and, if additional objects have been created during the CANopen slave configuration, the additional network variables for CANopen.

The following must always be observed here:

The standard network variables for Motion Control may be read by the user, but not written.

These are the standard Motion Control network variables for the setpoint values:

"u_controlword.."	for the axis control word
"ud_PosIpSetAngel.."	for the synchronized position angle setpoint value of the axis

These are the standard Motion Control network variables for the actual values:

"u_statusword.."	for the axis status word
"ud_Enc1ActAngle.."	for the synchronized position angle actual value of the axis
"si_modes_of_operation_display.."	for the axis operating mode

The additional network variables for CANopen are written (setpoint values) and read (actual values) by the user.

If during the configuration of the CANopen slave communication you have created the additional network variables (and their link to CANopen slaves) suggested in the sections [►PD Receive tab◄](#) from page 69 and [►PD Transmit tab◄](#) from page 71, the additional network variable for the setpoint values is:

"di\_ApplicationParam1.." for Application parameter 1 of the axis

The additional network variable for the actual values is:

"dj\_SpeedActValue.." for the actual speed value of the axis



#### NOTE

The standard network variables for Motion Control may be read by the user, but not written.

The additional network variables for CANopen are written (reference values) and read (actual values) by the user.



#### NOTE

You use the CANopen master module together with Motion Control.

No Motion Control axis (from the CANopen standpoint it is a network node) may be operated via the function blocks from the CANopen\_PLCC01\_30bd00 library (or higher) under ProProg wt III together with Motion Control.

This means a network node is either operated via the function blocks from the CANopen\_PLCC01\_30bd00 library (or higher) under ProProg wt III or by Motion Control.



#### NOTE

When using Motion Control, the initialization of the CANopen master module is assumed by Motion Control. The function block COM405\_KERNEL\_INIT from the CANopen\_PLCC01\_30bd00 library (or higher) under ProProg wt III may not be used.



---

**NOTE**

When using Motion Control, the evaluation of the network states and the network node states is assumed by Motion Control.

The COM405\_NETWORK\_CONTROL function block from the CANopen\_PLC01\_30bd00 library (or higher) may **not be used to control** the network states and the network node states.

The COM405\_NETWORK\_CONTROL function block from the CANopen\_PLC01\_30bd00 library (or higher) may be **used to evaluate** the network states and the network node states.

---



---

**NOTE**

When using Motion Control, the network is started up automatically by Motion Control.

---



---

**NOTE**

The network variables must be read and written in the Motion Control event task.

Exception: The PDO transmission type of the network variables is set asynchronous.

---

## 4.4 Programming data exchange with PROPROG wt II and CANop405\_PLC01\_20bd03 library

The data exchange with PROPROG wt II and the CANop405\_PLC01\_20bd03 library (or higher) can be programmed with

- the CANopen master module, software version < **01.20**  
(i.e.  $\leq$  BMC-M-ETH-02/CAN-04-01-00-001-**000**)
- the CANopen master module, software version  $\geq$  **01.20**  
(i.e.  $\geq$  BMC-M-ETH-02/CAN-04-01-00-001-**001**).

### 4.4.1 Overview

The following chapter describes the various possibilities for using a CANopen master module in a CANopen network in detail. You will be shown both the basic mechanisms as they are defined in the DS 301 profile and the related use of the function blocks from the CANop405\_PLC01\_20bd03 library (or higher) according to the DS 405 profile. The descriptions will be accompanied by a step-by-step example project for PROPROG wt II. In this example project, a simple network with a CANopen master module for the b maXX controller PLC and a CANopen slave module for the b maXX controller from the device series of the b maXX drives will be programmed. Please see the related operating instructions for information on commissioning the CANopen slave module. Please also observe any necessary settings of the b maXX controller from the device series of the b maXX drives to enable communication with the CANopen slave module.

#### NOTE



The resulting example project is intended to explain the CANopen functions possible with the CANopen master module in greater detail, and to make them easy to understand. It should not be considered a fully functional application. The data written to the CANopen slave in the examples have no function meaning with regard to their values. Therefore, make sure that any drive connected is not ready to operate!

Of course, we offer far more possibilities with the CANopen master module and the function blocks than can be described within this manual. Nor is it possible to cover all details of the function blocks here. Please see the related online help PROPROG wt II for details on the function blocks.

### 4.4.2 Steps to be carried out

To use the CANopen master module for data exchange in a CANopen network, the following steps must be carried out:

- 1 Physical commissioning of the CANopen network (see the operating instructions for the BMC-M-ETH-02/BMC-M-CAN-04 module and the operating instructions for the CANopen network nodes)
- 2 Creation of a PROPROG wt II project with the CANop405\_PLC01\_20bd03 library (or higher)

### 3 Implementation of the function blocks for CANopen communication, translation and transmission of the project

Of course, you can also begin with step 2 and carry out the physical commissioning in a later step.

#### 4.4.3 Commissioning CANopen network

---

For details on commissioning the CANopen network, please see the operating instructions on the BMC-M-ETH-02/BMC-M-CAN-04 module and the operating instructions of the other CANopen network nodes.

Following the physical commissioning of the network and the connection of the power supply of the b maXX system (also see the operating instructions of the power supply unit for the b maXX controller PLC), the CANopen master module is ready for operation after approx. 5 seconds.

H1 (green): Flashes H2 (red): Off	CANopen: The module waits for the initialization by an application program on the b maXX controller PLC
H1 (green): Off H2 (red): On	CANopen: The CAN is in the Bus-Off state

Flashes:  $t_{on} = 200 \text{ ms}$ ,  $t_{off} = 200 \text{ ms}$

All other states of the LEDs H1 and H2 indicate errors. For more information on this topic, please see the related operating instructions for the CANopen master module BMC-M-ETH-02/BMC-M-CAN-04.

#### 4.4.4 Creation of a project and integration of the CANop405\_PLC01\_20bd03 library

---

##### 4.4.4.1 Procedure when creating a project

---

To use the CANopen master module with the CANopen function blocks according to profile DS 405, you require a PROPROG wt II project for the b maXX controller PLC01. If you have not yet created your own project for your application, please create it with the template *BMC\_M\_PLC01*. To do this, you require a PROPROG wt II Version 3.1 from Build 274. The version number of PROPROG wt II is located on the sleeve of the installations CD of PROPROG wt II or in PROPROG wt II itself in the menu item ? \ *About*. Also check whether the BM\_TYPES\_20bd06 library (or higher) is present in your PROPROG wt II project. If this is not the case, please integrate this library in your project. It contains important data types for CANopen. Then integrate the CANop405\_PLC01\_20bd03 library in your project.

##### 4.4.4.2 Example: Creating the project "CANopenMaster\_C\_Example"

---

The example project "CANopenMaster\_C\_Example" was created with the template *BMC\_M\_PLC01* and integrated in the CANop405\_PLC01\_20bd03 library.

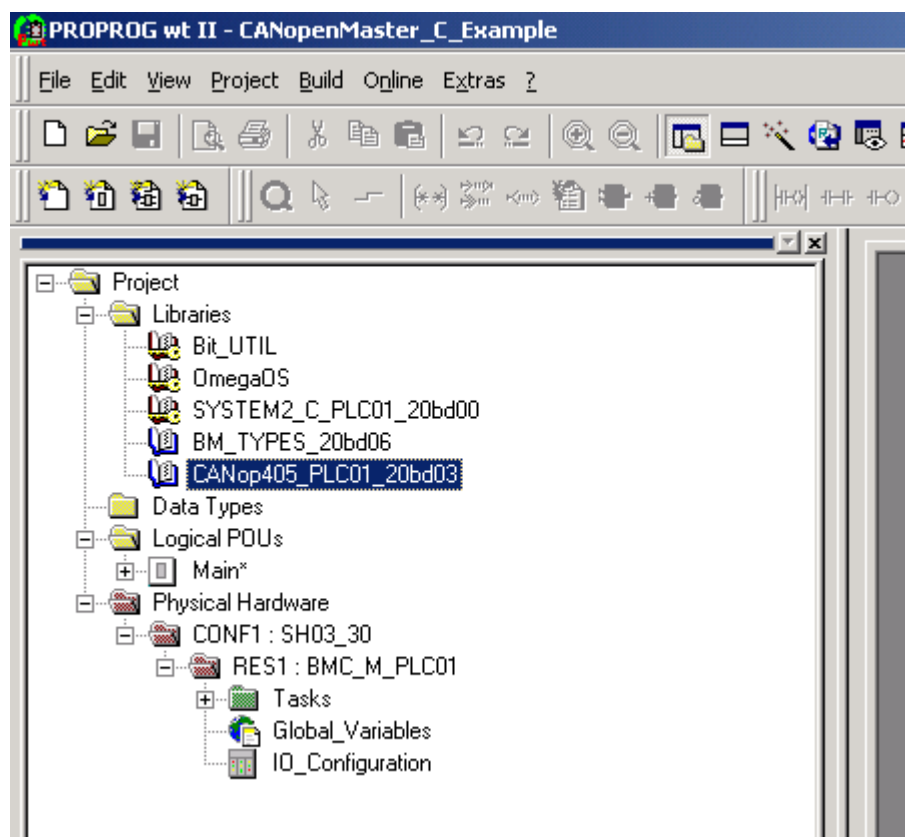


Figure 72: Example: Creating the project "CANopenMaster\_C\_Example"

#### 4.4.5 Creating a global variable for data exchange

A global variable is required for the data exchange between the BMC-M-ETH-02/BMC-M-CAN-04 module and the function blocks. It has no further meaning for the user. This global variable has already been created in your project if the project was created with the template *BMC\_M\_PLC01*. This global variable is connected to the function blocks for CANopen on the input/output *\_CANop405\_CTRL*.

Depending on the module number (01 to 05) of the module, the global variables *\_CANop405Ma\_Ctrl\_MNr\_01* to *\_CANop405Ma\_Ctrl\_MNr\_05* are available to you. These are also contained in the "Global\_Variables" worksheet:

```
Module CANopen-Master (BMC-M-ETH-02 / BMC-M-CAN-04) *)
_CANop405Ma_Ctrl_MNr_01      AT %MB3.1016384   : CANop405_PLC_BMSTRUCT;
_CANop405Ma_Ctrl_MNr_02      AT %MB3.2016384   : CANop405_PLC_BMSTRUCT;
_CANop405Ma_Ctrl_MNr_03      AT %MB3.3016384   : CANop405_PLC_BMSTRUCT;
_CANop405Ma_Ctrl_MNr_04      AT %MB3.4016384   : CANop405_PLC_BMSTRUCT;
_CANop405Ma_Ctrl_MNr_05      AT %MB3.5016384   : CANop405_PLC_BMSTRUCT;
```

Figure 73: Global variables for the CANopen master module are dependent on the module number

If the global variable required for the respective module number is not present in the project, then create the global variable *\_CANop405Ma\_Ctrl\_MNr\_01* (to *\_CANop405Ma\_Ctrl\_MNr\_05*) of the data type *CANop405\_PLC\_BMSTRUCT* depend-

ing on the module number (01 to 05). This variable must be declared as a global variable and placed on the basic address to the CANopen master communication of the BMC-M-ETH-02/BMC-M-CAN-04 module. The basic address is dependent on the module number (01 to 05) you have set on S1. The following module numbers are possible:

Module Number (S1)	Basic Address for CANopen Master Communication
01	%MB3.2016384
02	%MB3.3016384
03	%MB3.4016384
04	%MB3.5016384
05	%MB3.6016384
06 to 15	reserved

### 4.4.6 Initializing CANopen master module

---

#### 4.4.6.1 Procedure for initializing CANopen master module

---

The CANopen master module is initialized with the FB CANop405\_INIT. To use this function block, please proceed as follows:

- Create a POE with PLC type SH03-30 and processor type *BMC\_M\_PLC01*. This POE is to be run later in a cold-boot and warm-boot task.
- Position the FB CANop405\_INIT in this POE.
- Connect the function block with variables of the correct data type. Set the baud rate for the CANopen network on the input *us\_BAUDRATE*. The following values are possible:

us_BAUDRATE	Network Baud Rate
0	1 Mbit/s
1	800 kbits/s
2	500 kbits/s
3	250 kbits/s
4	125 kbits/s
5	100 kbits/s
6	50 kbits/s
7	20 kbits/s
8	10 kbits/s



#### 4.4.6.2 Example: Initializing CANopen master module

The connection of the FB CANop405\_INIT for an initialization with 500 kbits/s can be as follows:

(\* Initialise the CANopen-Master with 500 kBit/s \*)

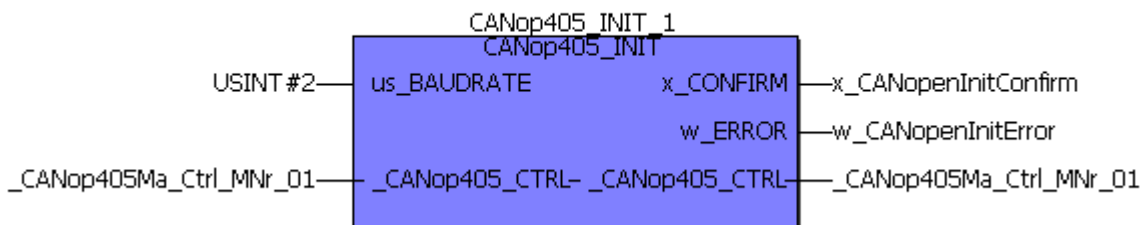


Figure 74: Initialization of CANopen master with FB CANop405\_INIT

- Create a task for the cold boot and the warm boot of the PLC if this is not yet present in your project. Integrate the created POE for the initialization of the module in the two tasks.
- Translate the project and load it as a boot project on the PLC. Switch the b maXX system off and then on again.

The FB CANop405\_INIT signals a successful initialization with  $x\_CONFIRM = 1$  and  $w\_ERROR = 16\#0$ .

Example:

(\* Initialise the CANopen-Master with 500 kBit/s \*)

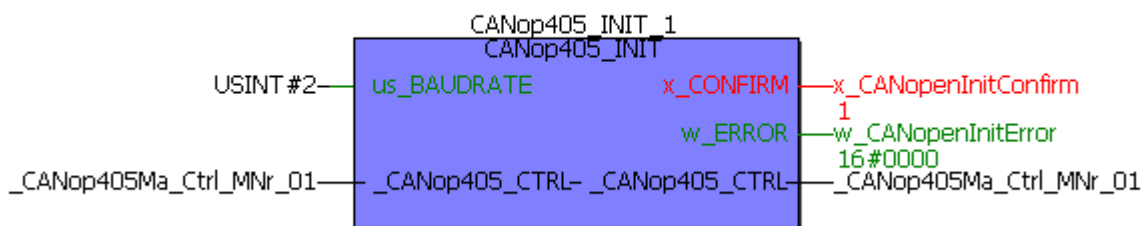


Figure 75: Initialization of CANopen master with FB CANop405\_INIT - Online

A successful initialization is also indicated by the LEDs on the CANopen socket on the module:

LED	Meaning
H1 (green): <b>On</b> H2 (red): <b>Off</b>	CANopen: The CANopen master module is initialized and ready for data transmission

Also set the correct baud rate for the other CANopen nodes. See the related documentation for information on doing this.

### 4.4.7 Starting individual network nodes - NMT

#### 4.4.7.1 Definition according to CANopen specification

The communication states of network nodes are controlled by means of NMT (Network Management) by a CANopen master. The following communication states of a device exist:

- INITIALIZATION
- PRE-OPERATIONAL
- STOPPED
- OPERATIONAL

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

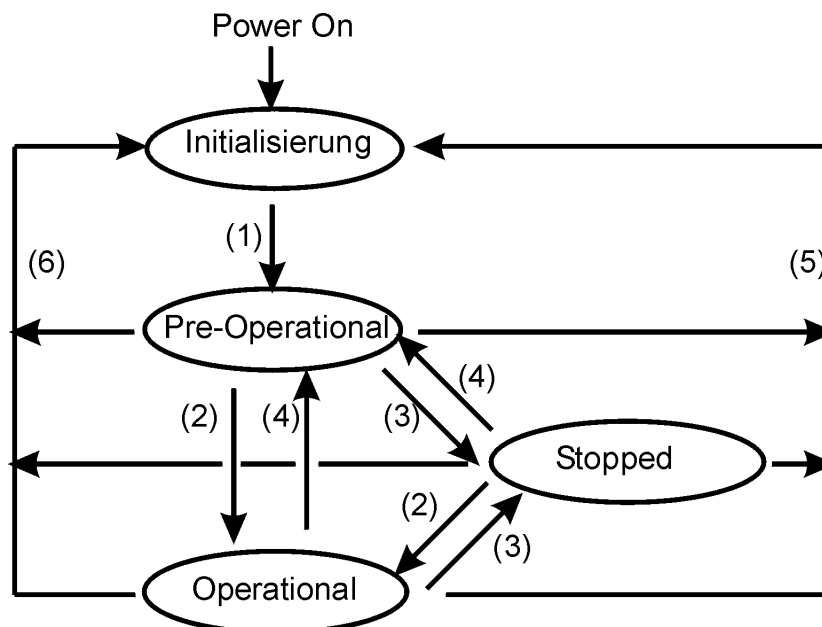


Figure 76: State transitions

Following the INITIALIZATION (triggered by switching on the network node), the PRE-OPERATIONAL state is automatically reached. If a network node is in this status, it can be configured via SDOs. Data exchange is not possible via PDOs.

In the STOPPED state, only Node Guarding is possible. Neither SDOs nor PDOs can be transmitted or received. There are six possible transitions between the individual states. The transition is either triggered automatically (Power On), by a command from the master or node internally (e.g. in case of an error).

- (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

The CANopen master module itself has no states or state transitions according to the CANopen definition.

The CANopen master can trigger the state transitions (2) to (6) with a telegram. This telegram is unconfirmed, i.e. the CANopen master has no information on whether a network node has received the telegram and whether it has carried out the state change. On the CANopen master the state of a network node can only be read out via Node Guarding (see the chapter [▶4.4.11 Monitoring network nodes with Node Guarding◀](#) from page 145 onward).



#### NOTE

Please observe the effects of the individual state transitions! A "Reset-Node" can trigger a reset of the entire network node, and therefore have undesirable effects on your application. Inform yourself on possible effects in the descriptions of the individual network nodes.

#### 4.4.7.2 Outputting NMT commands

NMT commands are output by the CANopen master with the FB CANop405\_NMT. To use this function block, please proceed as follows:

- Create a POE with PLC type SH03-30 and processor type *BMC\_M\_PLC01*. This POE is to be run later in a cyclical task.
- Position the FB CANop405\_NMT in this POE.
- Connect the function block to variables of the correct data type. Enter the number of the network node which is to evaluate the NMT command on the input *us\_DEVICE*. The command is evaluated by all network nodes with the value *USINT#0*. The command for the state transition is specified on *us\_TRANSITION\_STATE*. The individual values are assigned to the commands as follows:

<i>us_TRANSITION_STATE</i>	Command
1	Start Remote Node
2	Stop Remote Node
3	Enter Pre Operational
4	Reset Node
5	Reset Communication

- Create a cyclical task with a medium to low priority if such a task is not yet present in your project. Integrate the created POE in this task with the FB CANop405\_NMT.
- Translate the project and load it as a (boot) project on the PLC.
- Start the project.

## 4.4 Programming data exchange with PROLOG wt II and CANop405\_PLC01\_20bd03 library

The NMT command is output with `x_ENABLE = 1`. The FB `CANop405_NMT` signals a successful execution with `x_CONFIRM = 1` and `w_ERROR = 16#0`.

### 4.4.7.3 Example: Network management with NMT

To use the FB `CANop405_NMT`, we create a POE and position the function block in it. We then assign the created POE a cyclical task with a 200 ms call interval. We want to switch all network nodes into the OPERATIONAL state. `us_DEVICE` must therefore have the value 0 and `us_TRANSITION_STATE` the value 1. After connecting the function block, translate the project and transmit it as a project to the PLC. Start the project. The OPERATIONAL state is activated on all network nodes with `x_ENABLE = 1`:

```
(* Network control by FB CANop405_NMT *)
```

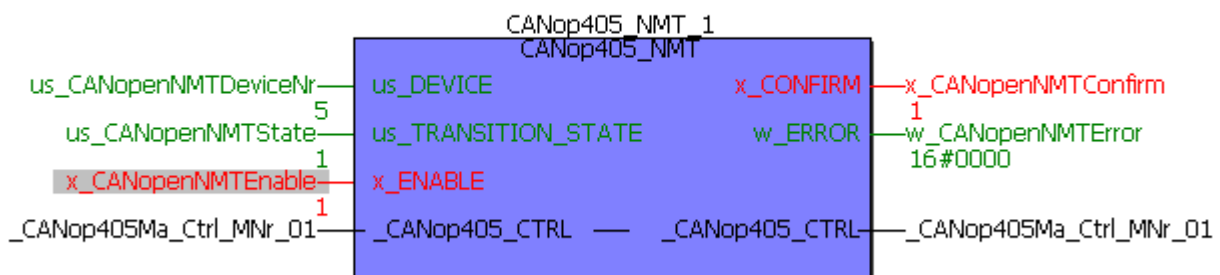


Figure 77: Network management with FB `CANop405_NMT`

## 4.4.8 Service data exchange with SDO

### 4.4.8.1 Definition according to CANopen specification

#### a) Objects of a network node

In case of access via SDOs (Service Data Objects), reading or writing accessing of the objects is always carried out. Depending on the profile, the objects can, for example, show configuration data, device functions or parameters. A network node has an object directory which contains all objects of the device. This object directory can be accessed directly via SDOs. The objects actually supported by a network node are contained in its documentation, as not all objects defined in a profile must be supported. In addition, manufacturer-specific objects can also be defined for a device. The three most important areas in an object directory of a device are:

- Objects of the communication profile (especially DS 301)
- Manufacturer-specific objects
- Objects of the device profile

Objects are always addressed via an index (16-bit) and a sub-index (8-bit). In addition to the index and the subindex, the object directory also contains information on the name, data type, attribute, default value and meaning of an object.

Examples of object entries are:

Object according to DS 301:

Index	Subindex	Name	Data Type	Attribute	Default Value	Meaning
1000h	00	Device Type	Unsigned 32	ro	402	Supported device profile

This entry in the object directory of the BM4-O-CAN-03 option module (CANopen slave for b maXX controller from the series of the b maXX drives) indicates that the object with the index 1000h and the subindex 0h contains the number of the supported device profile. The entry 402 therefore means that the device profile DSP 402 (drives) is supported. The value can only be read (ro = read only) and is transmitted as a 32-bit value. With an I/O terminal module, the same object would have the default value 401 for the device profile DS 401 (I/O modules).

Object according to DSP 402:

Index	Subindex	Name	Data Type	Attribute	Default Value	Meaning
6040h	00	Control word	Unsigned 16	rw	-	Control word

This entry in the object directory of the BM4-O-CAN-03 option module (CANopen slave for b maXX controller from the series of the b maXX drives) indicates that the object with the index 6040h and the subindex 0h represents the control word (parameter 300). The value can be read and written (rw = read write) and is transmitted as a 16-bit value. As this is a device profile-specific object, this has a different meaning for an I/O terminal module, i.e. the filter for digital inputs.

Manufacturer-specific object:

Index	Subindex	Name	Data Type	Attribute	Default Value	Meaning
41BAh	00	SVG set value 1	Signed 16	rw	16384	Setpoint value generator Setpoint value 1

This entry in the object directory of the BM4-O-CAN-03 option module (CANopen slave for b maXX controller from the series of the b maXX drives) indicates that the object with the index 41BAh and the subindex 0h represents the setpoint value 1 of the setpoint value generator (parameter 442). The value can be read and written (rw = read write) and is transmitted as a signed 16-bit value.

**NOTE**

The CANopen master module itself has no objects, neither communication profile-specific nor device profile-specific. With the CANopen master module and the function blocks from the CANop405\_PLC01\_20bd03 library, they can directly access the individual SDO and PDO telegrams.

---

## b) Sequence of SDO communication

An SDO communication is equivalent to the client/server communication model, i.e. the CANopen master module is the client and transmits a telegram with the order to a network node to apply data or transmit data. The network node acts as a server, applies the data and confirms this with a telegram or transmits the requested data. The opposite direction, i.e. the CANopen master replies as a server to the request of a client, is not possible. With the CANopen master, the "expedited" SDO transfer can be carried out according to DS 301 with up to 4 bytes of data per order. The "segmented" and the "Block" SDO transfer according to DS 301 are not supported.

**4.4.8.2 Writing objects with SDO**

---

Objects are written via SDO via the function blocks CANop405\_SDO1\_WRITE to CANop405\_SDO8\_WRITE. Up to 8 SDO orders can be started simultaneously with these function blocks. An FB is available for each SDO order.

**NOTE**

The function blocks CANop405\_SDO1\_WRITE to CANop405\_SDO8\_WRITE may not be instantiated several times.

---

To use the function blocks CANop405\_SDO1\_WRITE to CANop405\_SDO8\_WRITE, please proceed as follows:

- Create a POE with PLC type SH03-30 and processor type *BMC\_M\_PLC01*. This POE is to be run later in a cyclical task. You can also use the POE with the FB CANop405\_NMT.
- Position one or several of the FBs CANop405\_SDO1\_WRITE to CANop405\_SDO8\_WRITE in this POE as needed.
- Connect the function blocks to variables of the correct data type and assign them the desired values. On the input *us\_DEVICE*, specify the number of the network node you want to write an object to. *w\_INDEX* and *b\_SUBINDEX* are the addresses of the object, *us\_DATALENGTH* is the size of the object in bytes and the data to be written are specified on *d\_DATA*.
- Create a cyclical task with a medium to low priority if such a task is not yet present in your project. Integrate the created POE in this task. If you use the POE with the FB CANop405\_NMT, no further task need be created.
- Translate the project and load it as a (boot) project on the PLC. Start the PLC.

The object is transmitted via SDO with `x_ENABLE = 1`. The FBs `CANop405_SDO1_WRITE` to `CANop405_SDO8_WRITE` signal a successful execution with `x_CONFIRM = 1` and `w_ERROR = 16#0`. As the writing of objects via SDO is confirmed by other network nodes, you may receive an error message in `w_ERROR` and `ud_ERRORINFO`. One reason can, for example, be that the object cannot be written (attribute = `ro`). Please see the online documentation for the function blocks for details on the error messages.



#### NOTE

The same node number (`us_DEVICE`) may not be used simultaneously for the FBs `CANop405_SDOx_READ` (see next chapter) and the FBs `CANop405_SDOx_WRITE` (with `x = 1` to 8). The FBs `CANop405_SDOx_READ` and `CANop405_SDOx_WRITE` may not be active at the same time.

#### 4.4.8.3 Example: Writing object via SDO

We want to write the parameter 1172 "Ramp Function Generator Starting Time" of the `b maXX` controller (from the series of the `b maXX` drives) with 56 s. The parameter 1172 has the following object data in the object directory of the CANopen slave option module:

Index	Subindex	Name	Data Type	Attribute	Default Value	Meaning
604Fh	00h	vl_ramp_function_time	Unsigned 32	rw	0	Ramp function generator starting time

The CANopen slave option module is set to the module number 5 via its DIP switch. We use the FB `CANop405_SDO1_WRITE`. We implement the FB `CANop405_SDO1_WRITE` in the POE with the function block `CANop405_NMT`. After connecting the function block, the project must be translated and transmitted to the PLC as a project. Start the project. We start the data transmission via SDO with `x_ENABLE = 1`.

In the online depiction, the function block should have the following appearance after transmitting the SDO:

## 4.4 Programming data exchange with PROPROG wt II and CANop405\_PLC01\_20bd03 library

(\* Write object via SDO to device "us\_DeviceNumber" \*)

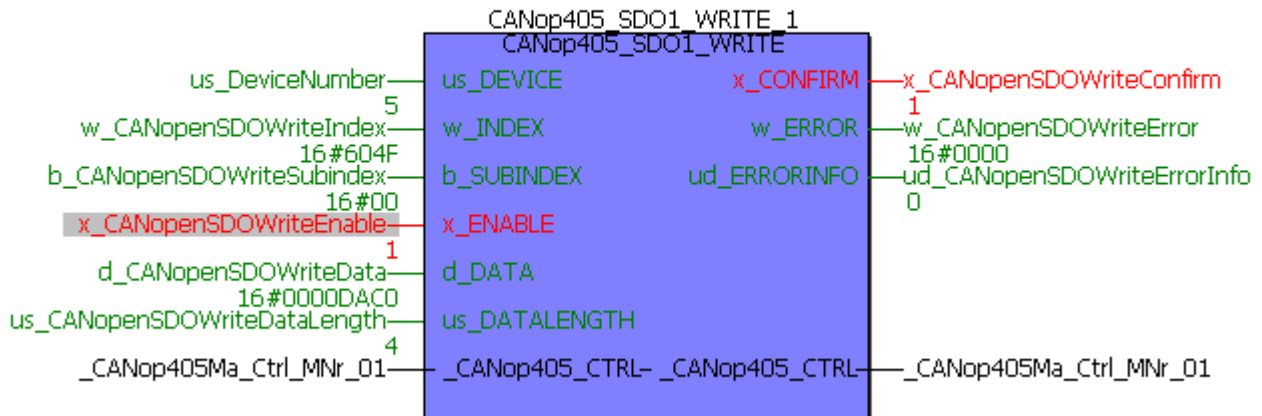


Figure 78: Writing the starting time with the FB CANop405\_SDO1\_WRITE

When checking the parameter 1172 "Ramp Function Time Starting Time" with WinBASS II, it should have the value 56 s:

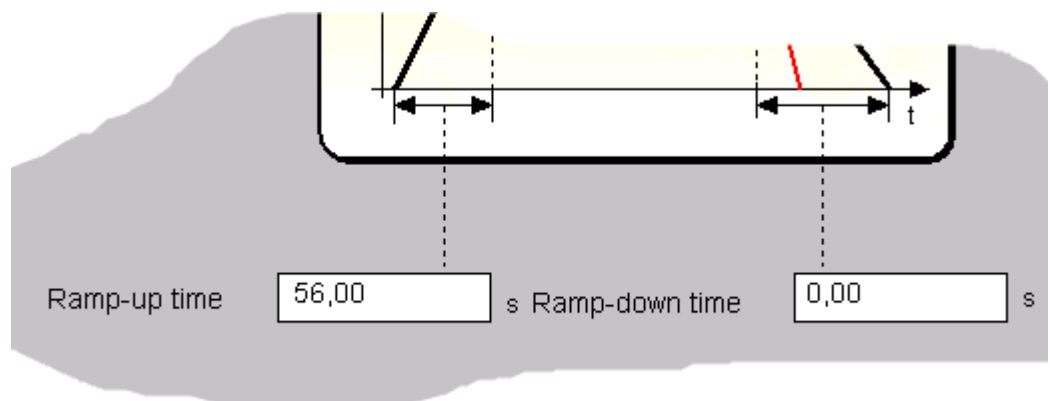


Figure 79: Starting time in WinBASS II

### 4.4.8.4 Reading objects with SDO

Objects are read via SDO via the function blocks CANop405\_SDO1\_READ to CANop405\_SDO8\_READ. Up to 8 SDO orders can be started simultaneously with these function blocks. An FB is available for each SDO order.



#### NOTE

The function blocks CANop405\_SDO1\_READ to CANop405\_SDO8\_READ may not be instantiated several times.



To use the function blocks CANop405\_SDO1\_READ to CANop405\_SDO8\_READ, please proceed as follows:

- Create a POE with PLC type SH03-30 and processor type *BMC\_M\_PLC01*. This POE is to be run later in a cyclical task. You can also use the POE with the FB CANop405\_NMT or the function blocks for writing objects via SDO.
- Position one or several of the FBs CANop405\_SDO1\_READ to CANop405\_SDO8\_READ in this POE as needed.
- Connect the function blocks to variables of the correct data type and assign them the desired values. On the input us\_DEVICE, specify the number of the network node from which you want to read an object. w\_INDEX and b\_SUBINDEX are the addresses of the object. The size of the object read in bytes is indicated on us\_DATALENGTH, and on d\_DATA the read data is shown.
- Create a cyclical task with a medium to low priority if such a task is not yet present in your project. Integrate the created POE in this task.
- Translate the project and load it as a (boot) project on the PLC. Start the PLC.

The object is requested from the slave via SDO with x\_ENABLE = 1. The FBs CANop405\_SDO1\_READ to CANop405\_SDO8\_READ signal a successful execution with x\_CONFIRM = 1 and w\_ERROR = 16#0. As the reading of objects via SDO is confirmed by the slave, you may receive an error message in w\_ERROR and ud\_ERRORINFO. One reason may, for example, be a non-existent object. Please see the online documentation for the function blocks for details on the error messages.

#### NOTE



The same node number (us\_DEVICE) may not be used simultaneously for the FBs CANop405\_SDOx\_READ and the FBs CANop405\_SDOx\_WRITE (see the chapter [▶4.4.8.2 Writing objects with SDO](#) from page 110 onward) (with x = 1 to 8). The FBs CANop405\_SDOx\_READ and CANop405\_SDOx\_WRITE may not be active at the same time.

#### 4.4.8.5 Example: Reading object via SDO

Now we want to read back the previously written parameter 1172 "Ramp Function Generator Starting Time" of the b maXX controller (from the series of the b maXX drives). If this parameter has not been overwritten by another page and the device has not been switched off, its value must be 56 s. The parameter 1172 has the following object data in the object directory of the CANopen slave option module:

Index	Subindex	Name	Data Type	Attribute	Default Value	Meaning
604Fh	00h	vl_ramp_function_time	Unsigned 32	rw	0	Ramp function generator starting time

The CANopen slave option module is set to the module number 5. We implement the FB CANop405\_SDO1\_READ in the same POE as the function block CANop405\_SDO1\_WRITE. After connecting the function block, translate the project and

## 4.4 Programming data exchange with PROPROG wt II and CANop405\_PLC01\_20bd03 library

transmit it as a project to the PLC. Start the project. We start the data transmission via SDO with `x_ENABLE = 1`. In the online depiction, the function block should have the following appearance after transmitting the SDO:

(\* Read object via SDO to device "us\_DeviceNumber" \*)

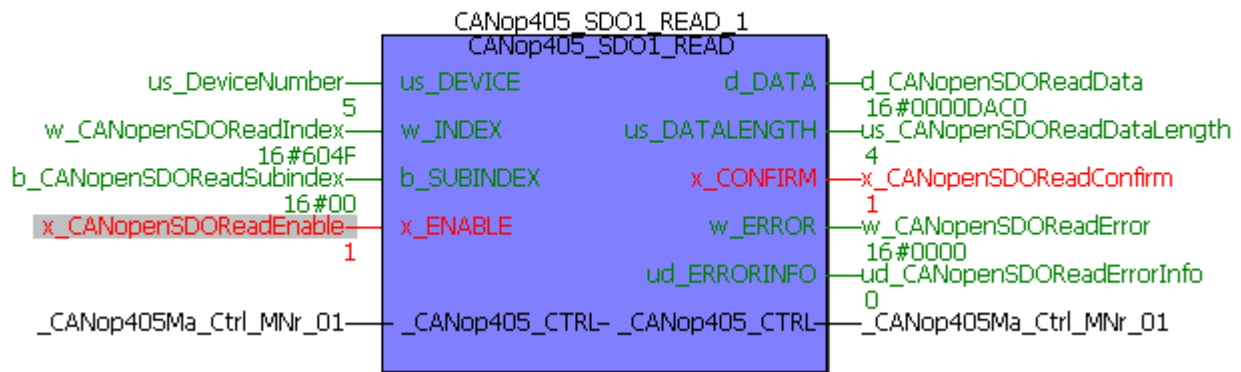


Figure 80: Reading the starting time with the FB `CANop405_SDO1_READ`

### 4.4.9 Process data exchange with PDOs

#### 4.4.9.1 Definition according to CANopen specification

A PDO communication (PDO = Process Data Object) is equivalent to the consumer/producer communication model, i.e. a network node generates a PDO telegram, which is processed by one or several other network nodes. There is no confirmation of the receipt of the telegram. The CANopen master can also generate and process PDO telegrams. The transmission itself must be configured. For example, you must in particular specify when the data are to be applied in a PDO, or at what point in time the data are to be entered in a PDO. In contrast to the data exchange via SDOs, an object of a network node is not directly accessed with PDOs. On the contrary, a network node is informed as to which objects with index, subindex and data length are contained in a PDO telegram. These objects are then taken from a PDO when it is written and entered in the PDO when it is read. Up to 8 bytes of data are available per telegram.

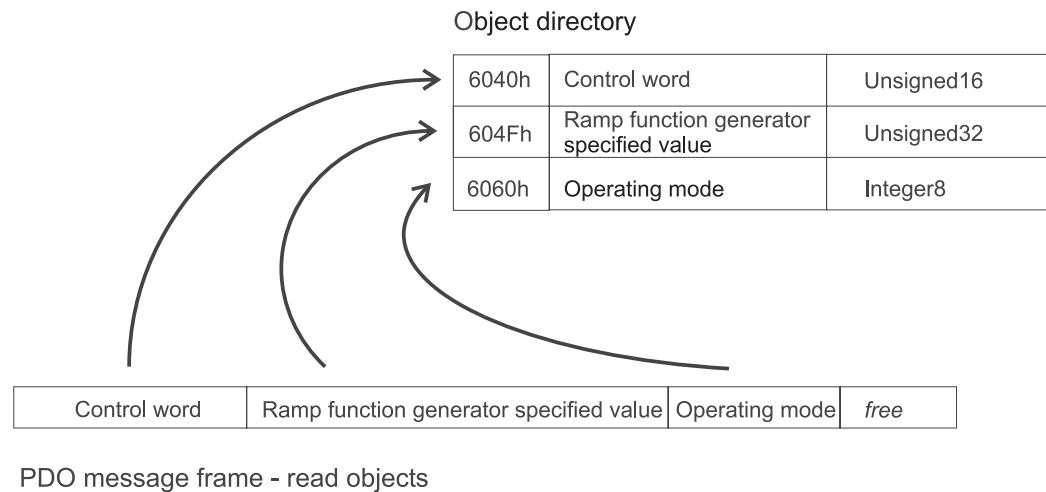


Figure 81: Reading objects from a PDO telegram

Only user data are contained in the PDO telegram itself. Information on the object index or subindex is no longer contained, as this is already known to the network node through the configuration carried out previously. This process of imaging objects in a PDO is also called "mapping".

Data exchange via PDOs can be divided into three phases:

- 1 Configuration of the transmission behavior of a PDO on each network node
- 2 Configuration of the PDO mapping on each network node
- 3 Cyclical data exchange via PDOs

The configuration of the PDOs is stored in communication profile-specific objects of a device. These objects are read and written via SDOs (see the chapter [▶4.4.8 Service data exchange with SDO](#) from page 108 onward). PDOs may only be configured in the PRE-OPERATIONAL state of a network node. Details on the configuration objects are explained in the following.

#### 4.4.9.2 Configuration of transmission properties of a PDO

The properties for the transmission of a PDO are described by communication profile-specific objects of a network node. In the following the configuration object for the first receive PDO (1400h) and the first transmit PDO (1800h) are each described for this purpose. To configure all other PDOs, the next highest object index must be used, i.e. the second receive PDO is configured via the object 1401h, the third receive PDO via the object 1402h etc. For the transmit PDOs, these are the objects 1801h, 1802h etc. A maximum of 512 receive PDOs and 512 transmit PDOs are possible. The values of the objects have the following meaning according to the DS 301 communication profile:

## 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	Transmission type of the PDO: synchronous or asynchronous
Subindex 3	<i>Not in use</i>
Subindex 4	Entry for compatibility. Optional.
Subindex 5	<i>Not in use.</i>

## 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	Transmission type of the PDO: synchronous, asynchronous, event-controlled.
Subindex 3	Transmission delay time of the PDO. Optional.
Subindex 4	Entry for compatibility. Optional.
Subindex 5	Timer value if the PDO transmission time is controlled. Optional.

Let's have a closer look at the meaning of the subindices.

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

The Subindex 0 indicates the number of the last valid subindex. The value is at least 2 and increases in accordance with the other entries (subindices) to be supported optionally. The value is generally "read only", i.e. is only of interest for information purposes.

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

The Subindex 1 contains various information. The value has the data type Unsigned32 and is divided up 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 the chapter <a href="#">▶4.2.2.3 Data exchange and objects of physical bus system◀</a> from page 53 onward), only 4 PDOs are possible for writing and 4 PDOs for reading. If a device supports more than these 4 PDOs, then you must generally 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 the value 1 and is not set to 0 by the user until it has assigned a COB-ID.
30	0: the PDO can be requested via a remote request 1: the PDO cannot be requested via a remote request
29	0: 11-bit CAN ID 1: 29-bit CAN ID (is not supported by the CANopen master module)
28 - 11	is not supported by the CANopen master module
10 - 0 (LSB)	COB-ID of the PDO. In the default settings of a device, the first 4 PDOs for writing and reading have the COB-IDs according to the predefined identifier assignment (see the chapter <a href="#">▶4.2.2.3 Data exchange and objects of physical bus system◀</a> from page 53 onward)

If you do not require more than 4 PDOs for writing and 4 PDOs for reading, then this value is usually only of interest to you for information purposes.

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

This value is very important for you, as it specifies when a PDO is transferred. The type of transmission is configured in Subindex 2. Here the following assignment applies for transmission and receive PDOs:

Subindex 2/ Type	Effect Transmit PDO 1800h	Receive PDO 1400h
0 Synchronous	Transmission is carried out after each received SYNC telegram if an event has occurred <sup>2)</sup> (see the chapter <a href="#">▶4.4.10 Synchronizing data exchange◀</a> from page 140 onward)	The PDO with matching COB-ID received before the last SYNC telegram is applied
1 - 240 Synchronous	Transmission is carried out after reception of the set number of SYNC telegrams	The PDO with matching COB-ID received before the last SYNC telegram is applied
252 Remote	When a SYNC telegram is received, the PDO is updated. Transmission is not carried out until after a remote request is received.	<i>Not possible</i>

Subindex 2/ Type	Effect Transmit PDO 1800h	Receive PDO 1400h
253 Remote	An update and transmission are carried out after a remote request is received.	<i>Not possible</i>
254 Asynchronous	Transmission is carried out manufacturer-specific. Remark: For this type, most devices transfer the PDO when a time even occurs (expiration of a timer). The time is set under Subindex 5. <sup>1)</sup>	Each PDO with a matching COB-ID is applied on reception
255 Asynchronous	Transmission is carried out in dependence on the supported device profile. Remark: With this type, most devices transmit the PDO as soon as one of the mapped values has changed. <sup>2)</sup>	Each PDO with a matching COB-ID is applied on reception

- 1) Time-controlled transmission means the transmission condition is linked to a timer. This timer is set for the first transmit PDO with Subindex 5 in the object 1800h (16-bit). The resolution is milliseconds. The timer is started during the state change to OPERATIONAL. The PDO is then transmitted cyclically with the time set in the timer. The timer is deleted by writing the value "0" to Subindex 5. Time-controlled reception does not exist. Instead, all PDOs are applied with a matching COB-ID.
- 2) Event-controlled transmission means the transmission condition is linked to the change in a value of one of the mapped objects. For example, if 3 objects are mapped (status word, actual speed value, actual operating mode), the PDO is transmitted as soon as at least one of the three values changes. If the values remain constant, no PDO is transmitted. This enables the bus load to be reduced (telegrams are only transmitted if they contain new information). Event-controlled reception means that all PDOs are applied with a matching COB-ID.

#### NOTE



Please note that some devices only have certain transmission types implemented and may also have different default settings. Should you not receive a PDO from a network node, or should a network node not apply a PDO, this is frequently due to the transmission type setting.

#### Object 1400h/1800h Subindex 3 - Transmission delay time of the PDO

With the Subindex 3 a transmission delay can be set which does not extend the reaction time with the relative first value change, but is active for changes which directly follow this change. The inhibit time (transmission delay time) describes the minimum time period which must be waited between the transmission of two identical telegrams. 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-controlled PDO transmission

The value of Subindex 5 specifies the time in ms for the transmission of the PDO (see Subindex 2).

#### 4.4.9.3 Mapping objects in a PDO

As a second configuration step for data exchange via PDOs, the mapping of the objects in a PDO must be specified. For mapping as well, there are special communication profile-specific objects in which this is described for a network node. In the following the configuration object for the first receive PDO (1600h) and the first transmit PDO (1A00h) are each described for this. For the configuration of all other PDOs, the next higher object index must be used, i.e. the mapping of the second receive PDO is configured via the object 1601h, the third receive PDO via the object 1602h etc. The objects have the following detailed meaning according to DS 301:

##### 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

Let's have a closer look at the meaning of the subindices.

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

Subindex 0 indicates the number of objects mapped in this PDO. As soon as you want to change the mapping of a PDO, you must first set the number of mapped objects to zero. Then you can change the mapping and subsequently set the number of mapped objects in Subindex 0. Please note that here device-dependent restrictions may exist for the number of objects to be mapped.

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

Subindices 1 to 64 reflect 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 content has the following meaning:

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

The objects are mapped consecutively in a PDO.

Please note that not every object can be mapped in a PDO.



#### NOTE

Default mappings are defined in the device profiles. Should the default mapping values be sufficient for you, you need not carry out any further settings on the network nodes here. The appendix contains an overview of the default mappings for devices according to profile DS 401 and DSP 402.

---

#### 4.4.9.4 Cyclical data exchange with PDO

Following the configuration of the transmission properties and the mapping of the PDO, the actual data exchange via PDOs can begin. The CANopen master module transmits PDOs to which the network nodes react in accordance with their setting. The CANopen master can also receive PDOs transmitted by the network nodes. The data exchange via PDOs is always only possible in the OPERATIONAL state of the network nodes. The PDOs cannot be configured in the OPERATIONAL state. Please note that the data exchange via PDOs must always also be viewed from the standpoint of the network nodes, i.e. if the CANopen master transmits a PDO, then this is also a receive PDO (RxPDO) on the master side, as the network node receives this PDO. The same also applies to a transmit PDO (TxPDO) received by the master.

---

#### 4.4.9.5 Writing objects via PDOs

a) Mechanism on CANopen master module

Objects are written via PDOs for the CANopen master module with the FB CANop405\_PDO\_WRITE. The mechanism which runs in the process is as follows:





the CANopen with 1 Mbit/s, you can transmit a calculated 7 PDOs with 8 bytes of data each per ms. However, this requires that no other data traffic takes place. But this is very improbable, as you probably also want to receive PDOs and use other CANopen functions like Node Guarding and Sync. This means the actual maximum possible bus load is largely determined by your application. As a result, fewer than 7 PDOs can actually be transmitted per ms.

Repeat these steps for all PDOs you want to write.

Program the configuration and carry out the following:

- Initialize the transmission of PDOs with the FB CANop405\_PDO\_INIT. You must position this function block before the FB CANop405\_INIT. The number of the highest transmission mailbox used must be specified on the FB CANop405\_PDO\_INIT. A maximum of 40 transmission mailboxes can be used.
- Set the network node in the PRE-OPERATIONAL state via the FB CANop405\_NMT.
- Use the function blocks CANop405\_SDO1\_WRITE to CANop405\_SDO8\_WRITE described in the chapter "Write objects with SDO" to check and set the configuration data (objects 1400h, 1401h etc. and 1600, 1601 etc.) of the network node.
- To determine the COB-ID of the PDO, you can use the function block CANop405\_COB\_ID. This FB calculates the COB-ID from the account number and PDO number you determine. The FB should be used in an initialization task.

c) Example: Transmit configuration PDO

We want to cyclically write the ramp function generator input P1171 and the speed additional setpoint P1040 des of the BM4-O-CAN-03 option module (on the b maXX controller from the device series of the b maXX drives) every 2 ms. The receive PDO 2 of the BM4-O-CAN-03 option module is to be used. The PDO is to be applied when it arrives. As up until now no PDO has been set up for transmission, we will use Transmission mailbox 1. After all, we still know the node number from the previous chapters: 5. For the parameters 1171 and 1040 we still require data from the objective directory to configure the mapping:

Parameter	Index	Subindex	Data length
1171 Ramp Function Generator Input	6042	0	16 bits
1040 Speed Additional Setpoint	4410	0	32 bits

With P1171 the value to be written according to the object directory directly shows the speed in rpm. The standardization of P1040 via CANopen is: -100% to +100% rpm is equivalent to -230 to +230.

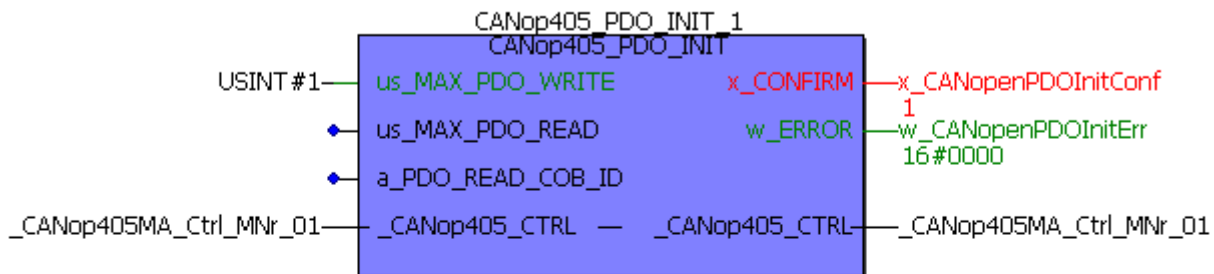
### NOTE



The parameters P1171 and P1040 are only to be written as examples and a change in the values is to be visible in WinBASS II. There is no functionality behind the written values; the values change suddenly in some cases. To avoid damage, you must therefore ensure that any drive which may be connected to the b maXX 4400 is not ready for operation!

First the FB CANop405\_PDO\_INIT must be configured. For this purpose, this FB is positioned in the initialization POE before the FB CANop405\_INIT. We specify the number of the last transmission mailbox used on us\_MAX\_PDO\_WRITE. As we only use the first transmission mailbox, this is USINT#1. After translating and downloading the project, the FB should have the following appearance in the online mode:

(\* Initialise the PDO for the CANopen-Master \*)



(\* Initialise the CANopen-Master with 500 kBit/s \*)

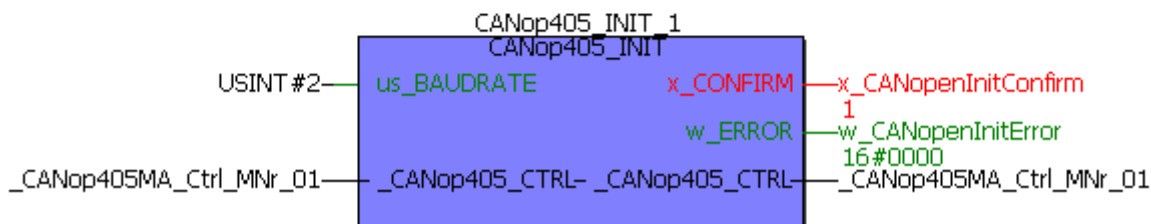


Figure 83: Initializing PDO transmission with the FB CANop405\_PDO\_INIT

The BM4-O-CAN-03 option module must first be brought into the PRE-OPERATIONAL state to carry out the configuration of the PDO. For this purpose, we use the FB CANop405\_NMT again. Immediately after switching on the b maXX 4400 device (with the BM4-O-CAN-03), this would not have to be carried out, as then the option module automatically assumes the PRE-OPERATIONAL state.

(\* Network control by FB CANop405\_NMT \*)

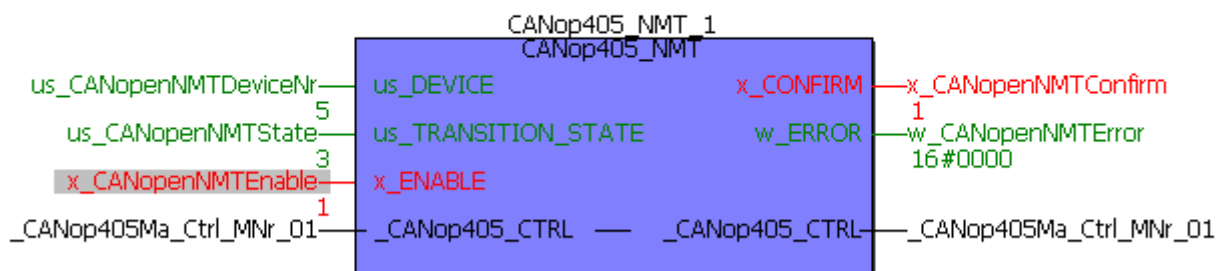


Figure 84: Switching CANopen slave into PRE-OPERATIONAL state

## 4.4 Programming data exchange with PROLOG wt II and CANop405\_PLC01\_20bd03 library

We now determine the COB-ID for the PDO via the FB CANop405\_COB\_ID. We implement this block in the existing initialization task. Node number 5 is connected to us\_DEVICE and node number 2 to us\_PDO\_NR (the second receive PDO is to be written). As this PDO is received by the BM4-O-CAN-03 option mode, i.e. is an RxPDO, we must connect the input x\_RX with 1. We output the COB-ID on a global variable. After translating and transmitting the project, we receive the value 305 h for the COB-ID in the online display.

(\* compute the COB-IDs for the PDOs to write \*)

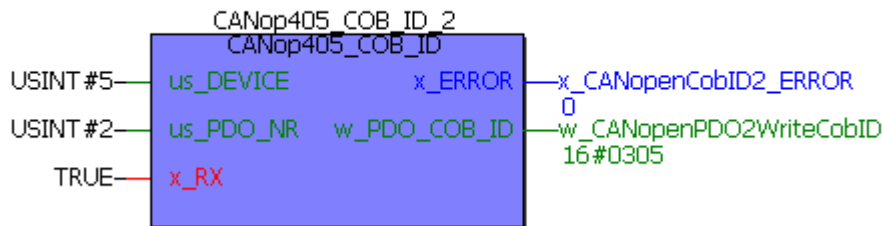


Figure 85: Determining COB-ID for second receive PDO with FB CANop405\_COB\_ID

Now we check the settings for this PDO on the BM4-O-CAN-03 option module. The first receive PDO is configured in the object 1400h for communication. According to this, we find the configuration for the second receive PDO in the object 1401h. We use the already existing FB CANop405\_SDO1\_READ, however with the connection for the object 1401h. This time we use variables and not constants to simply switch over to the next object on-line.

(\* Read object via SDO to device "us\_DeviceNumber" \*)

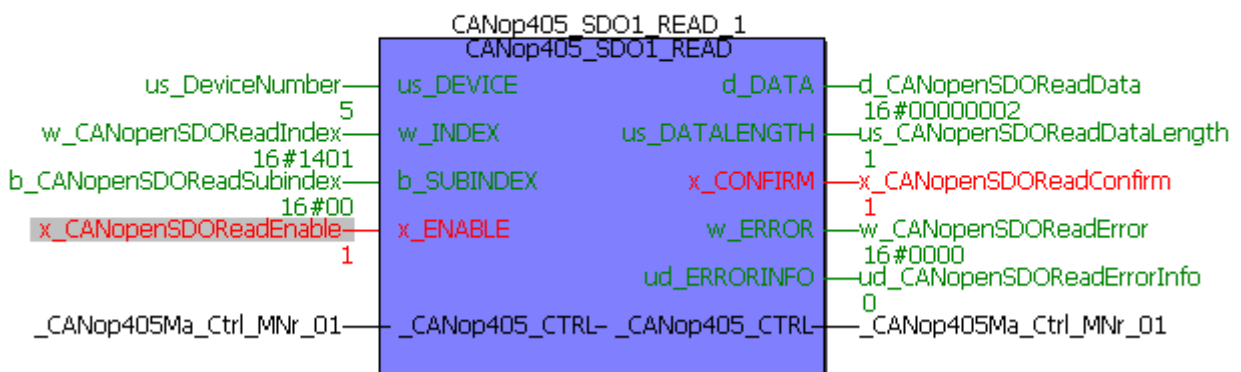


Figure 86: Reading out configuration of second receive PDO of CANopen slave - Subindex 0

For Subindex 0 we receive the value 2. This means the object 1401h has valid entries in Subindices 0 - 2. As we are interested in the set COB-ID and the validity of the PDO (both Subindex 1), as well as the transmission type (Subindex 2), we must read Subindex 1 and 2. We receive:

(\* Read object via SDO to device "us\_DeviceNumber" \*)

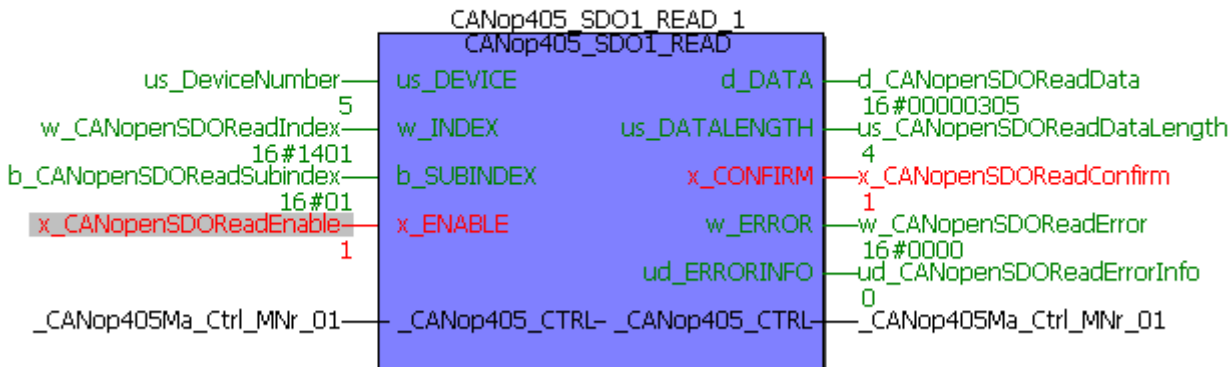


Figure 87: Reading out configuration of second receive PDO of CANopen slave - Subindex 1

The value 305h means that the COB-ID of this PDO 305h is, an 11-bit CAN-ID is used and the PDO is valid. This means the value of Subindex 1 is correct.

(\* Read object via SDO to device "us\_DeviceNumber" \*)

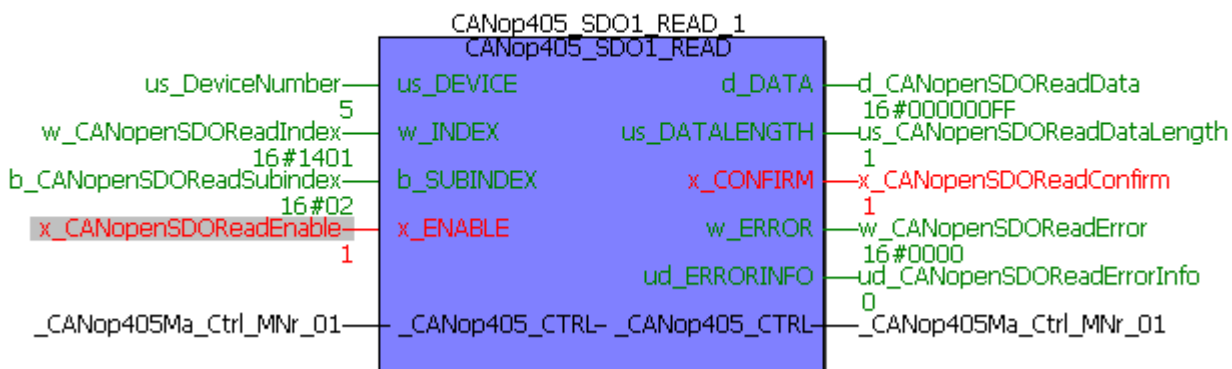


Figure 88: Reading out configuration of second receive PDO of CANopen slave - Subindex 2

The value FFh means that the transmission type for this PDO is 255, i.e. each PDO with a matching COB-ID (305h) is applied during reception. This value is also correct. If you receive other values for the object contents here, then you must change these with one of the FBs CANop405\_SDOx\_WRITE (x = 1 to 8).

Now we check the mapping for the second receive PDO in Object 1601h. We want to map the parameter 1171 first. For the entries in 1601h, we should therefore receive the following values taking the object index, subindex and data length into account:

Object 1601h

Subindex 0: 2h	Two mapped values
Subindex 1: 6042 00 10h	Index, Subindex, Length for P1171
Subindex 2: 4410 00 20h	Index, Subindex, Length for P1040

Check the values of Object 1601h according to the above method with the FB CANop405\_SDO1\_READ. In the example project we receive the following values:

Object 1601h

Subindex 0: 2h	Two mapped values
Subindex 1: 6040 00 10h	
Subindex 2: 6060 00 08h	

Except for Subindex 0, none of the values is correct. We must change the mapping. To do this we use the FB CANop405\_SDO1\_WRITE. In place of the constants, we connect global variables again to switch over to the next object easily online.

First we delete the mapping of the second receive PDO by setting the value of Subindex 0 to zero.

(\* Write object via SDO to device "us\_DeviceNumber" \*)

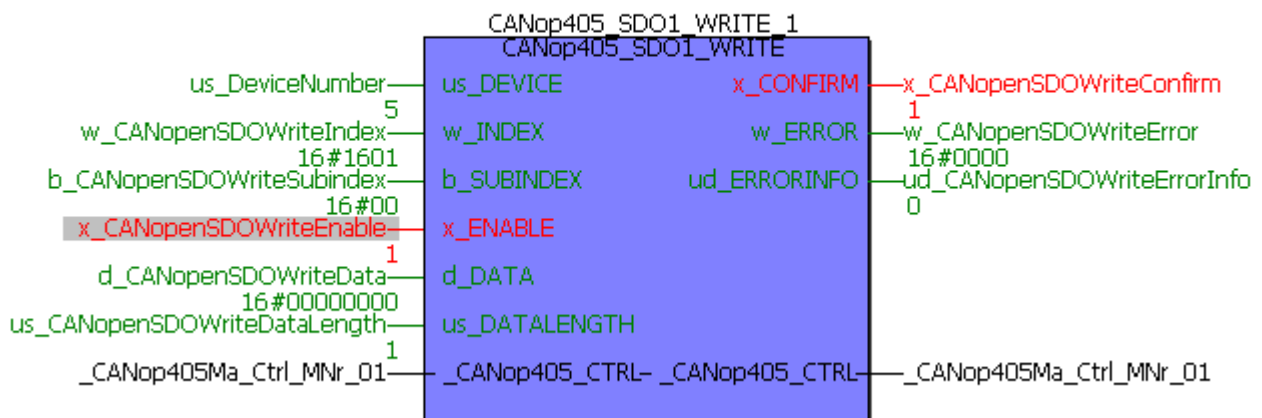


Figure 89: Deleting mapping of second receive PDO

Now we enter the mapping of the first Object P1171 in Subindex 1:

(\* Write object via SDO to device "us\_DeviceNumber" \*)

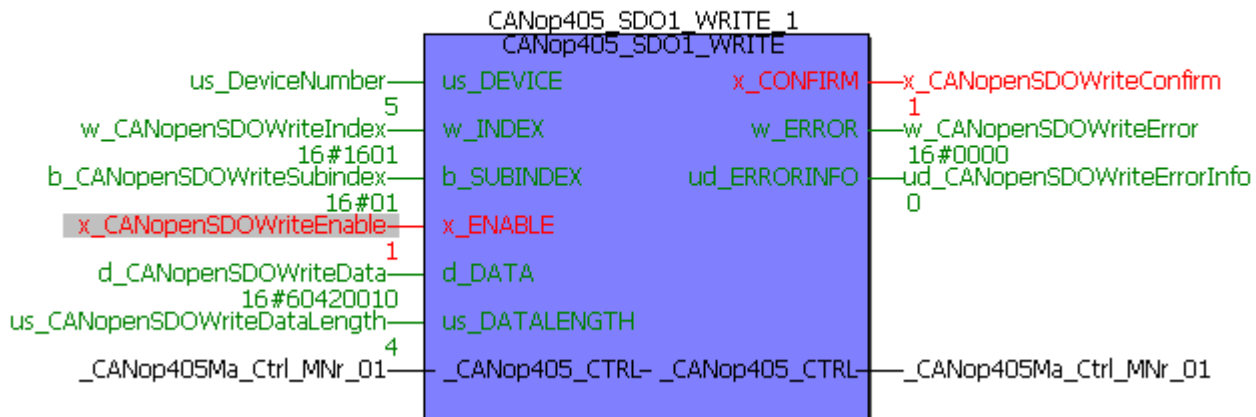


Figure 90: Entering first mapped object of second receive PDO

and then mapping of Object P1040 in Subindex 2:

(\* Write object via SDO to device "us\_DeviceNumber" \*)



Figure 91: Entering second mapped object of second receive PDO

Now all objects have been entered and we must release the mapping again by entering the number of mapped objects in Subindex 0:

## 4.4 Programming data exchange with PROLOG wt II and CANop405\_PLC01\_20bd03 library

(\* Write object via SDO to device "us\_DeviceNumber" \*)

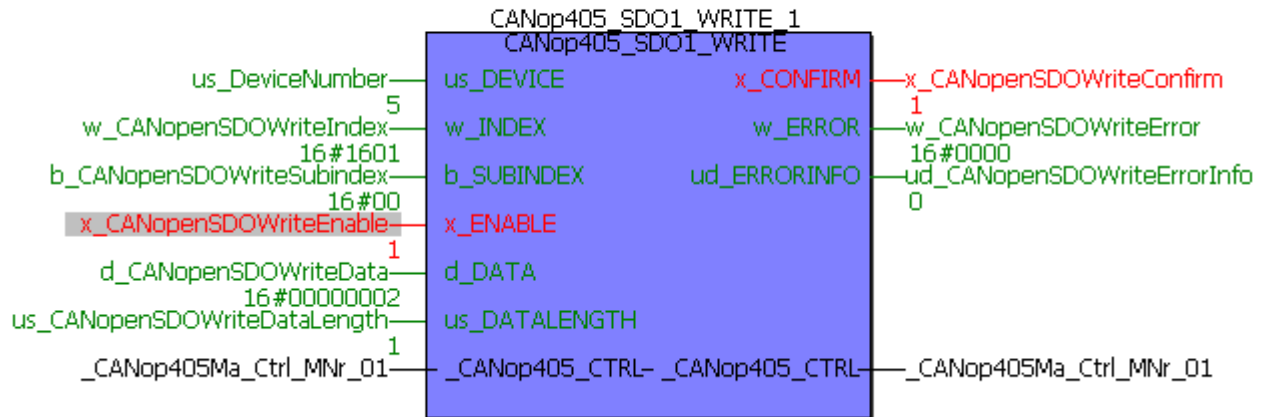


Figure 92: Entering number of mapped objects of second receive PDO

The BM4-O-CAN-03 option module offers the option of saving the set communication parameters. The next time the b maXX 4400 device is switched off/on, the stored communication parameters will then automatically be applied. Saving is triggered by writing the object 1010h, Subindex 2 with the value Wert 65766173h. We want to do this now:

(\* Write object via SDO to device "us\_DeviceNumber" \*)

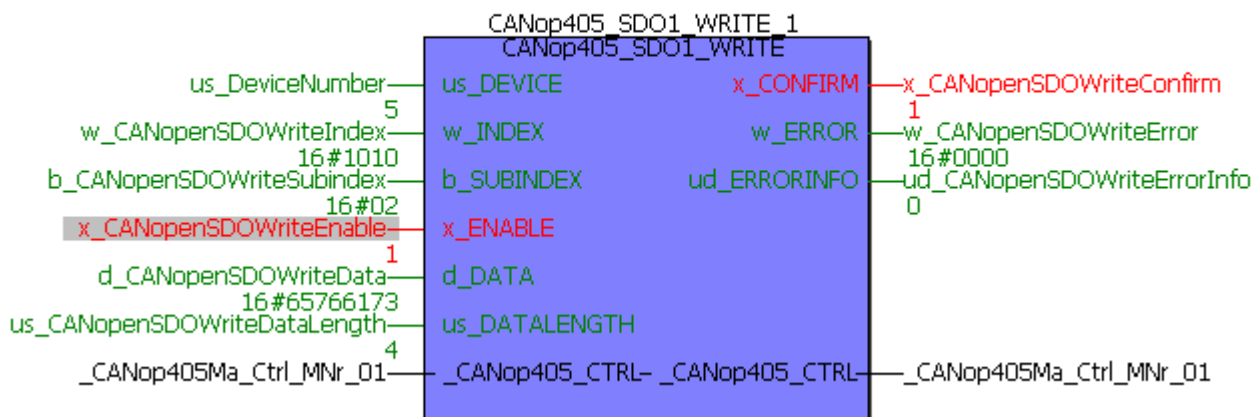


Figure 93: Saving communication parameters of CANopen slave

The second receive PDO on the BM4-O-CAN-03 option module is now completely configured and we can move on to the actual transmission of the PDO.

### d) Transmitting a PDO to a network node

A PDO is transmitted via the instances of the function block CANop405\_PDO\_WRITE. Up to 40 PDO orders can be started simultaneously with these function blocks. Here the number 40 matches the number of transmission mailboxes on the CANopen master mod-



ule which are available for this purpose. A transmission mailbox may not be used twice. However, it is possible to use a transmission mailbox consecutively for different PDOs. For a network node to apply a PDO, it must be in the OPERATIONAL state.

To use this function block CANop405\_PDO\_WRITE, please proceed as follows:

- Create a POE with PLC type SH03-30 and processor type *BMC\_M\_PLC01*. This POE is to be run later in a cyclical IRP task.
- Position one or several of the FBs CANop405\_PDO\_WRITE in this POE as needed.
- Connect the function blocks to variables of the correct data type and assign them the desired values. On the input *us\_PDO\_NR* you specify the number of the transmission mailbox you have selected. *w\_PDO\_COB\_ID* is the COB-ID of the PDO you want to write. *us\_DATALENGTH* designates the valid bytes within the PDO. It is the sum of all bytes of the mapped objects. The data for the objects are connected to *d\_DATA0* and *d\_DATA1*. If you want to request a remote PDO, set the input *x\_REMOTE* to 1.
- Create a cyclical bypass IRP task if such a task is not yet present in your project. With low time requirements for the PDO, you can also use a simple cyclical task with a priority of your choice. Integrate the created POE in this task.
- Translate the project and load it as a (boot) project on the PLC. Start the PLC.
- If necessary, carry out the configuration of the network nodes and set these to the OPERATIONAL state.

The PDO is transmitted with *x\_ENABLE* = 1. The FB CANop405\_PDO\_WRITE signals a successful execution with *x\_CONFIRM* = 1 and *w\_ERROR* = 16#0000. As the writing of a PDO is not confirmed by the slave, *x\_CONFIRM* = 1 only means that the PDO has been transmitted by the CANopen master module, but not whether it was also applied by the slave.

#### e) Example: Transmit PDO

We continue the example. The second receive PDO of the BM4-O-CAN-03 option module is configured and we can move on to the actual data exchange via PDOs. First the network node must be switched into the OPERATIONAL state. For this purpose, we use the FB CANop405\_NMT again:

(\* Network control by FB CANop405\_NMT \*)

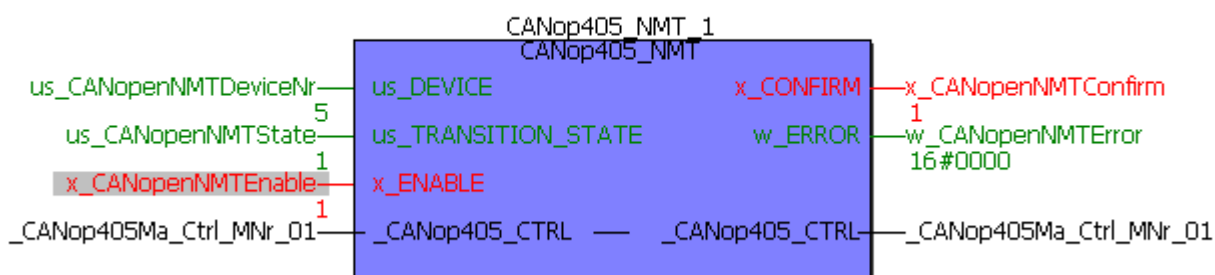


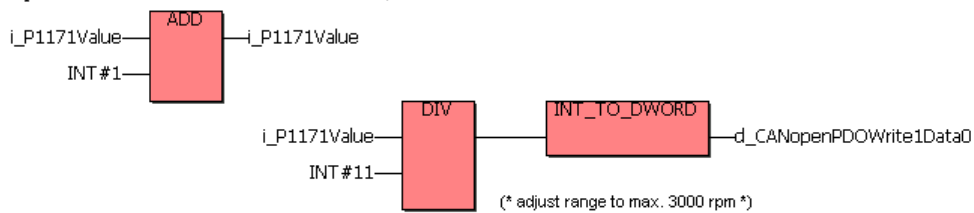
Figure 94: Switching CANopen slave into OPERATIONAL state

Then we create a POE for later integration in an IRP task. We place the FB CANop405\_PDO\_WRITE in this POE and connect it to corresponding constants and var-

## 4.4 Programming data exchange with PROPROG wt II and CANop405\_PLC01\_20bd03 library

ables. As previously specified, the first transmission mailbox is to be used, i.e. us\_PDO\_NR is = 1. We get the COB-ID from the FB CANop405\_COB\_ID from the initialization task via the global variable w\_CANopenPDO1WriteCobID. The data length of P1171 is two bytes and of P1040 four bytes. According to this, us\_DATALENGTH is given the value USINT#6. Due to mapping, P1171 lies in the first word of d\_DATA0 and P1040 in the second work of d\_DATA0, as well as in the first word of d\_DATA1. For recognizable data traffic to take place, both parameters are simply to be counted up or counted down, whereby P1171 is also limited to a maximum of 3,000 rpm. The finished code is then as follows:

(\* Compute P1171 for device 5\*)



(\* Compute P1040 for device 5\*)

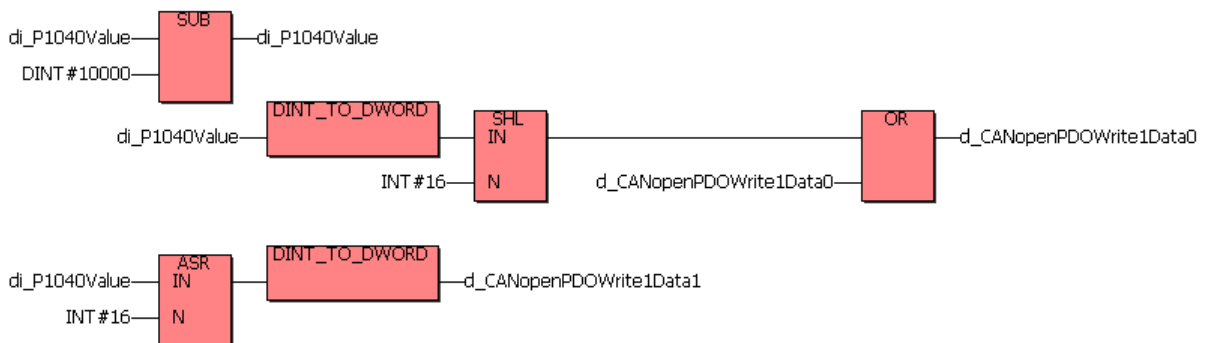


Figure 95: Changing data to be written

(\* Send PDO to device 5 \*)

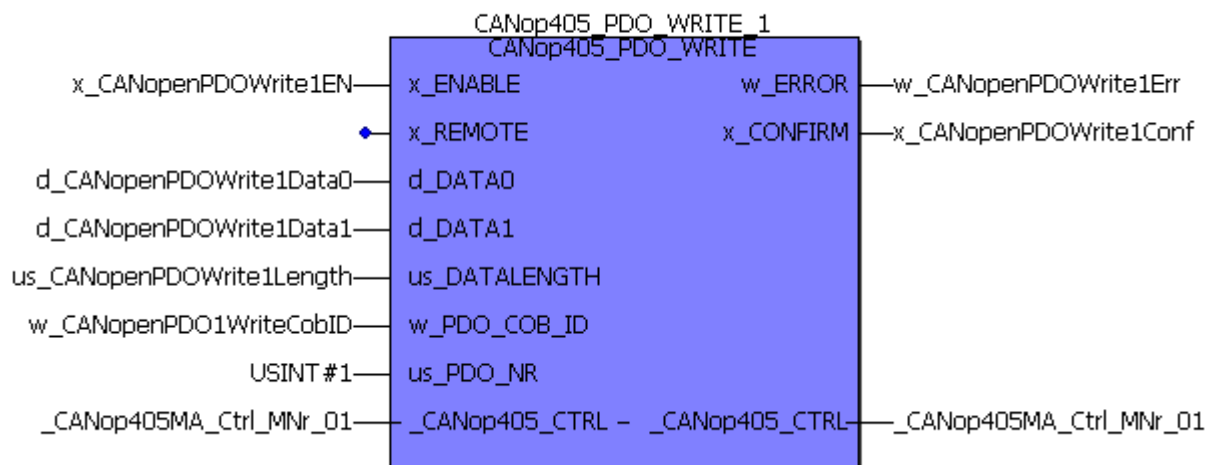


Figure 96: Writing data with FB CANop405\_PDO\_WRITE

To call the FB CANop405\_PDO\_WRITE every 2 ms, we also require a timer IRP task. For its configuration we require the FB INTR\_SET from the SYSTEM2\_C\_PLC01\_20bd00 library (or higher). We configure the FB INTR\_SET for a 2 ms CPU Timer 1 Interrupt and implement this function block in the CANopen initialization POE before the FB CANop405\_PDO\_INIT:

```
(* Initialise the Timer-IRP with 2ms *)
```

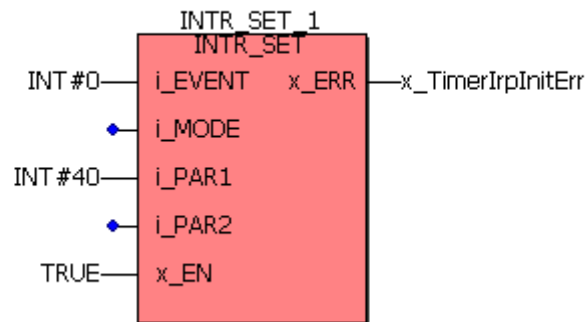


Figure 97: Setting up 2 ms timer IRP task for data transmission

Now the corresponding event task can be set up with the event "CPU-Timer 1" and bypass operation, and this event task assigned the POE with the FB CANop405\_PDO\_WRITE. Translate the project and transmit it to the PLC. Start the project.

In the CANopen slave (b maXX controller with BM4-O-CAN-03), the PDO configuration (set and saved above) is active and the CANopen slave is in the OPERATIONAL state.

If you have switched off the CANopen slave, the stored configuration will be loaded when it is switched on again, i.e. the PDO configuration (saved above) will be used. However, the CANopen slave is in the PRE-OPERATIONAL state after being switched on, which means the CANopen slave must be switched into the OPERATIONAL state by the CANopen master via the network management (see [▶Example: Network management with NMT](#) on Page 108).

The data transmission can be started with `x_ENABLE = 1` on the FB CANop405\_PDO\_WRITE:

## 4.4 Programming data exchange with PROPROG wt II and CANop405\_PLC01\_20bd03 library

(\* Send PDO to device 5 \*)

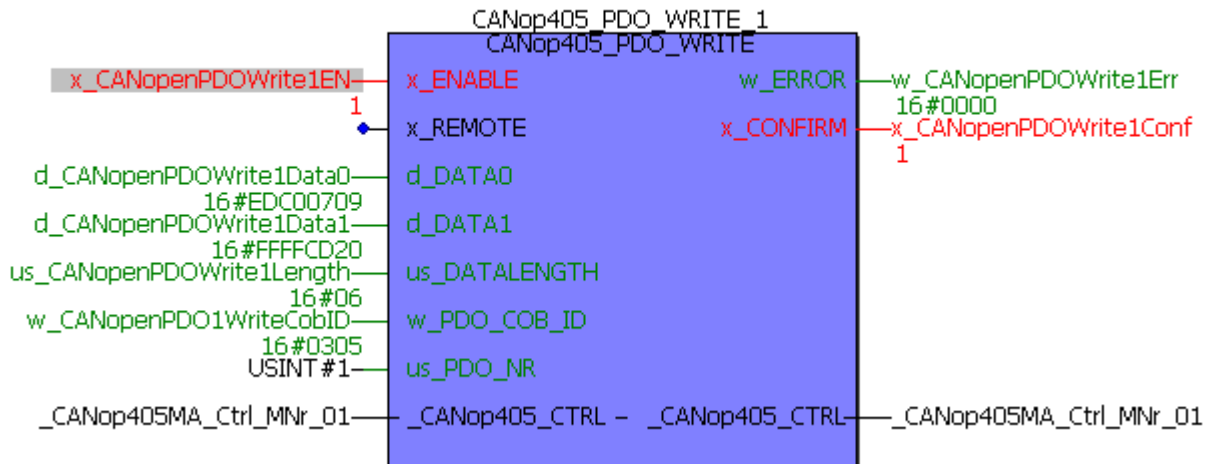


Figure 98: Writing data with FB CANop405\_PDO\_WRITE - Online

A correct transmission of the PDO is confirmed with  $x\_CONFIRM = 1$ . In WinBASS II you should be able to detect a change in the parameters 1171 and 1040.

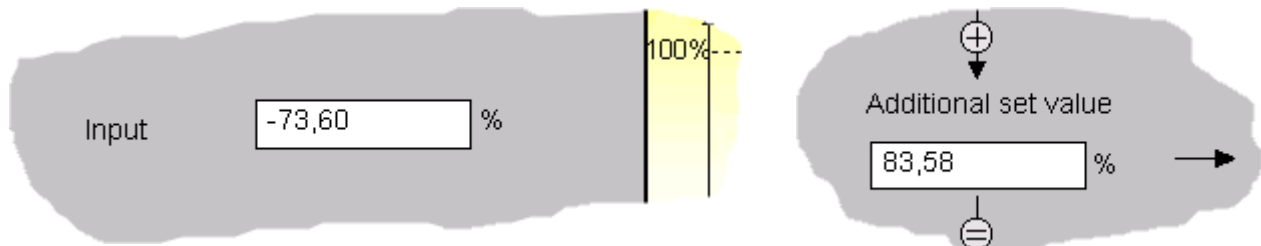


Figure 99: Written data in WinBASS II

Please note that these are "snapshots" in the online mode and may differ from your values.

### 4.4.9.6 Reading objects via PDOs

a) Mechanism on CANopen master module

Objects are read via PDOs for the CANopen master module with the FB CANop405\_PDO\_READ. The mechanism which runs in the process is as follows:

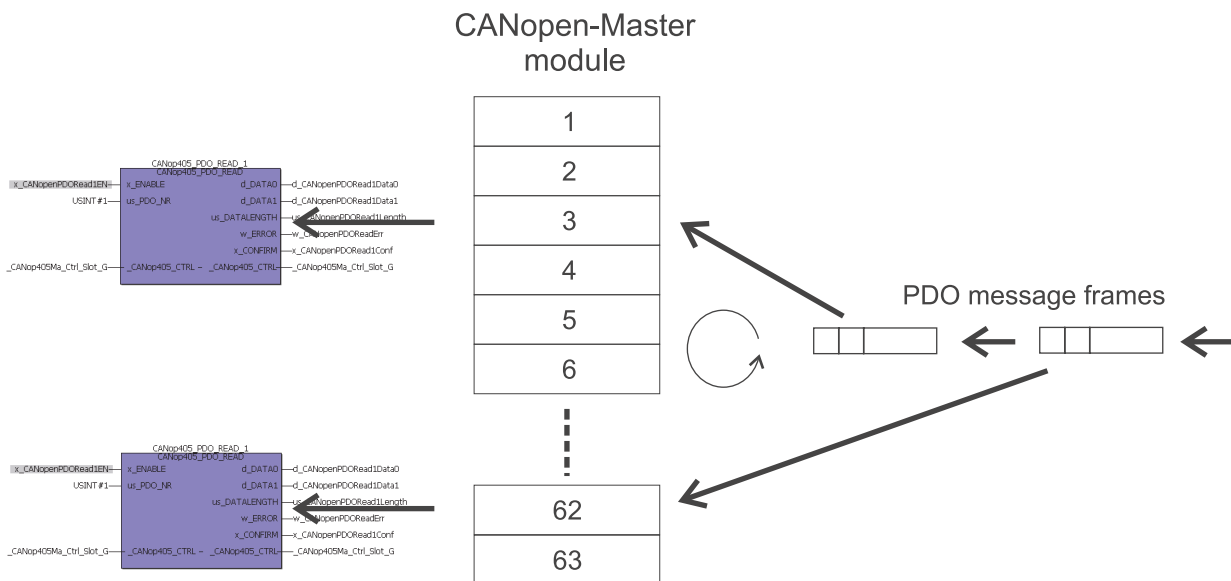


Figure 100: Reading mechanism on module for PDO

Each PDO to be received is assigned an FB CANop405\_PDO\_READ by corresponding instances being formed by this function block. Each of these function blocks must be assigned a reception mailbox on the CANopen master module via an input on the function block. A total of 63 reception mailboxes are available. This means that 63 reception orders for PDOs can be evaluated simultaneously. When a PDO arrives, the module enters it in a reception mailbox. The correct reception mailbox is selected by the module via the COB-ID of the PDO. For this purpose, the module must be informed during the initialization phase as to which COB-ID is assigned to which reception mailbox. This is carried out with the FB CANop405\_PDO\_INIT. A reconfiguration is no longer possible during operation. During the call, the FB CANop405\_PDO\_READ checks whether an entry is present in its assigned reception mailbox.

#### NOTE

Received PDOs may be lost if more PDOs arrive than are read out by the FB CANop405\_PDO\_READ, or if the FB CANop405\_PDO\_READ is currently applying a PDO from the reception mailbox. To reduce possible losses, the FB CANop405\_PDO\_READ should have a higher call interval than data can arrive. In addition, mechanisms for the controlled PDO transfer (e.g. SYNC or remote request, see corresponding chapter) should be used.

#### b) Preparation and configuration

Before you can receive a PDO, you must make several preparations and configuration settings on the network node which is to transmit the PDO. Proceed as follows for this purpose:

Specify the important data:



- Specify the number of the network node in the CANopen network which is to transmit the PDO.
- Specify which PDO of the network node is to be transmitted. To do this, see the documentation on the network node to determine how many PDOs the node supports.
- Specify a not yet assigned reception mailbox for the PDO. They should begin with reception mailbox 1 and assign the next reception mailboxes without gaps.
- Define the transmission type of the PDO. When PDOs are transmitted by a network node, you have various options. Inform yourself on these again in the chapter [► Configuration of transmission properties of a PDO◄](#) on Page 115, and in the documentation for the network node.
- Now specify the objects to be mapped in the PDO. Watch the default mapping of the network node. You may be able to use it for your application.
- Specify the time frame in which the PDO is to be evaluated. You create a task in PROPROG wt II later in accordance with the time frame to check the receipt of the PDO in the reception mailbox. The time frame should be adapted to the transmission type of the PDO. Please observe the maximum CANopen bus load. When operating the CANopen with 1 Mbit/s, you can receive a calculated 7 PDOs with 8 bytes of data each per ms. However, this requires that no other data traffic takes place. But this is very improbable, as you probably also want to transmit PDOs and use other CANopen functions like Node Guarding and Sync. This means the actual maximum possible bus load is largely determined by your application. As a result, fewer than 7 PDOs can actually be received per ms.

Repeat these steps for all PDOs you want to receive.

Program the configuration and carry out the following:

- Initialize the reception of PDOs with the FB CANop405\_PDO\_INIT. You must position this function block before the FB CANop405\_INIT. The number of the highest reception mailbox used must be specified on the FB CANop405\_PDO\_INIT. A maximum of 63 reception mailboxes can be used. You also carry out the assignment of the COB-IDs of the PDOs to the reception mailboxes on this FB. For this purpose an array is used. Its respective index point corresponds to a reception mailbox and its content is the COB-ID. You can enter values up to a maximum of Index 63. An error message is generated for entries which exceed this value and at Index 0.
- To determine the COB-ID of the PDO, you can use the function block CANop405\_COB\_ID. This FB calculates the COB-ID from the account number and PDO number you determine. The FB is therefore called before the FB CANop405\_PDO\_INIT.
- Set the network node in the PRE-OPERATIONAL state via the FB CANop405\_NMT.
- Use the function blocks CANop405\_SDO1\_WRITE to CANop405\_SDO8\_WRITE described in the chapter "Write objects with SDO" to check and set the configuration data (objects 1800h, 1801h etc. and 1A00, 1A01 etc.) of the network node.

c) Example: Receiving configuration PDO

The BM4-O-CAN-03 option module (on the b maXX controller from the device series b maXX drives) is to transmit the status word P301 and the speed additional setpoint P1040 every 5 ms. We want to use the transmit PDO 1 of the BM4-O-CAN-03 option module. On the side of the CANopen master module, the reception of the PDO is to be checked every 2 ms. As up until now no PDO has been set up for reception, we will use

reception mailbox 1. After all, we still know the node number: 5. For the parameters 301 and 1040 we still require a few data from the objective directory to configure the mapping:

Parameter	Index	Subindex	Data length
301 Status word	6041	0	16 bits
1040 Speed Additional Setpoint	4410	0	32 bits

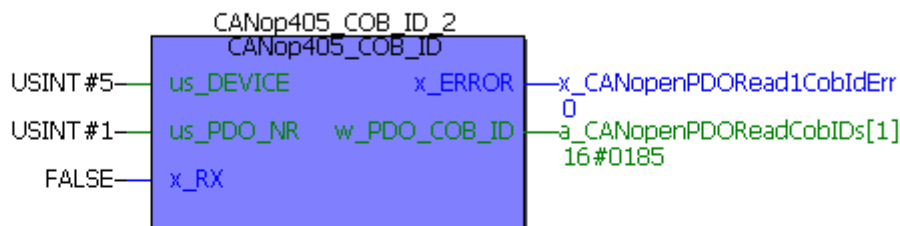
The standardization of P1040 via CANopen is:  $-100\%$  to  $+100\%$  rpm is equivalent to  $-2^{30}$  to  $+2^{30}$ .

First the FB CANop405\_PDO\_INIT must be configured. This FB is already present in the initialization POE due to the configuration of transmission. We specify the number of the last reception mailbox used on us\_MAX\_PDO\_READ. As we only use the first reception mailbox, this is USINT#1. The reception mailbox 1 must also still be assigned a COB-ID. The assignment is made via the input a\_PDO\_READ\_COB\_ID. Here an array of the data type WORD\_64\_BMARRAY must be connected which contains the COB-ID. As we use reception mailbox 1, the COB-ID must be entered on Index 1 of the array. We use the FB CANop405\_COB\_ID for the generation of the COB-ID. The node number 5 is connected to us\_DEVICE and node number 1 to us\_PDO\_NR. As this PDO is transmitted by the BM4-O-CAN-03 option mode, i.e. is an TxPDO, we must connect the input x\_RX with 0.

After translating and downloading the project, the initialization should have the following appearance in the online mode:

(\* Initialise the PDO for the CANopen-Master \*)

(\* Fill the array with the COB-IDs for reading PDOs \*)



(\* Initialise the PDOs \*)

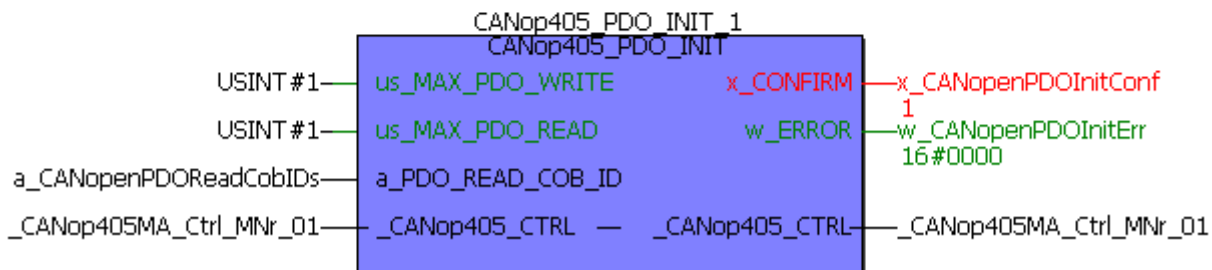


Figure 101: Initializing PDO transmission with the FB CANop405\_PDO\_INIT

For the COB-ID we receive the value 185h. Set the BM4-O-CAN-03 option module to the PRE-OPERATIONAL state with the FB CANop405\_NMT to carry out the configuration of the PDO.

Now we check the settings for this PDO on the BM4-O-CAN-03 option module. The first transmit PDO is configured in the object 1800h for communication. We use the already existent FB CANop405\_SDO1\_READ, however with the connection for the object 1800h.

(\* Read object via SDO to device "us\_DeviceNumber" \*)

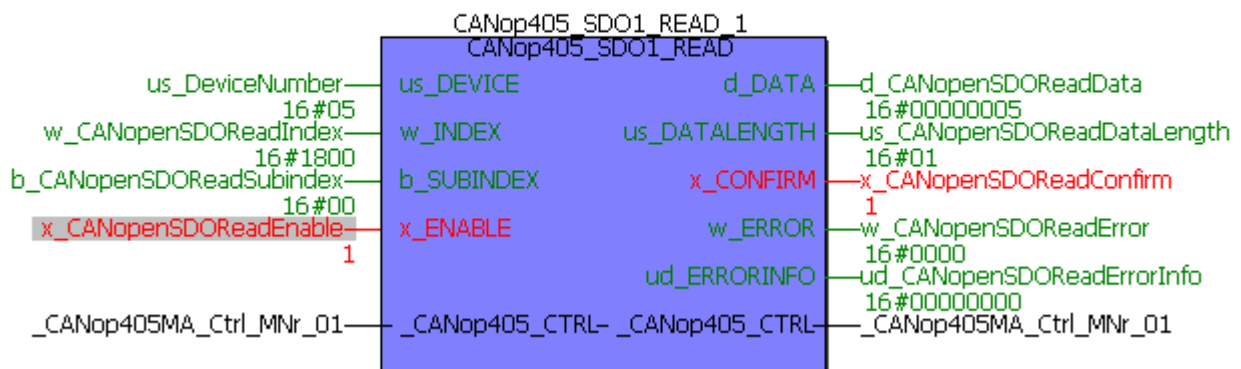


Figure 102: Reading out configuration of first transmit PDO of CANopen slave - Subindex 0

For Subindex 0 we receive the value 5. This means the object 1800h has valid entries in Subindices 0 - 5. As we are interested in the set COB-ID and the validity of the PDO (both Subindex 1), as well as the transmission type (Subindex 2) and the transmission time (Subindex 5), we must read Subindex 1, 2 and 5. We receive the following values according to the same method as for Subindex 0:

Object 1800h

Subindex 0:	5h	Five valid subindex entries
Subindex 1:	185h	COB-ID of first transmit PDO
Subindex 2:	FFh	Transmission type 255
Subindex 5:	0h	Timer value for transmission time

The value for the COB-ID of this PDO is 185h, an 11-bit CAN-ID is used and the PDO is valid. This means the value of Subindex 1 is correct. The value FFh of Subindex 2 means that the transmission type for this PDO is 255, i.e. the PDO will be transmitted in case of a value change of a mapped object. We must change this value to 254 with one of the FBs CANop405\_SDO1\_WRITE to CANop405\_SDO8\_WRITE, as we require a time-event controlled transmission:



(\* Write object via SDO to device "us\_DeviceNumber" \*)



Figure 103: Writing configuration of first transmit PDO of CANopen slave - Subindex 2

Subindex 5 contains the timer value for the time event. 0h is entered, however we want to transmit every 5 ms. This means Subindex 0 must be changed to the value 5 ms:

(\* Write object via SDO to device "us\_DeviceNumber" \*)

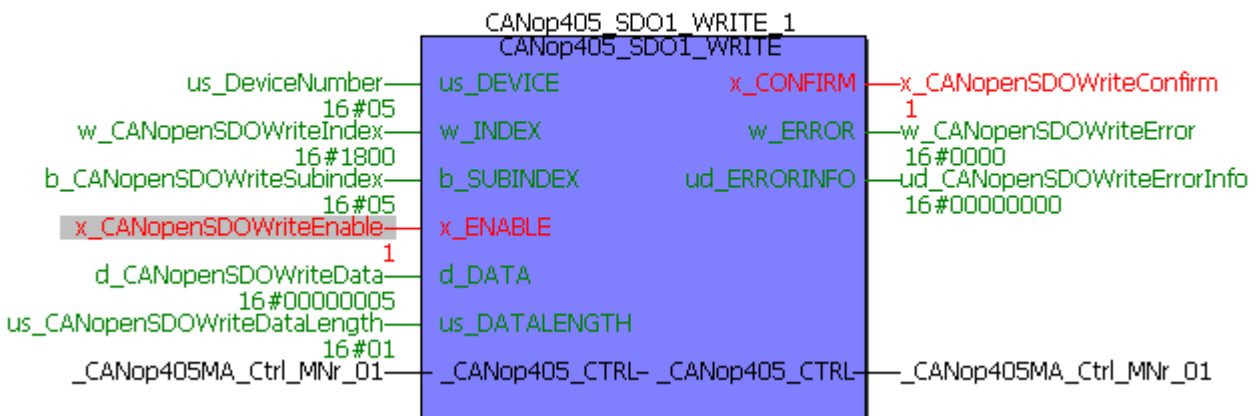


Figure 104: Writing configuration of first transmit PDO of CANopen slave - Subindex 5

Now we check the mapping for the first transmit PDO in Object 1A00h. We want to map the parameter 301 first. For the entries in 1A00h, we should therefore receive the following values taking the object index, subindex and data length into account:

Object 1A00h

Subindex 0:	2h	Two mapped values
Subindex 1:	6041 00 10h	Index, Subindex, Length for P301
Subindex 2:	4410 00 20h	Index, Subindex, Length for P1040



which are available for this purpose. A reception mailbox may not be used twice. For a network node to transmit a PDO, it must be in the OPERATIONAL state.

To use this function block CANop405\_PDO\_READ, please proceed as follows:

- Create a POE with PLC type SH03-30 and processor type *BMC\_M\_PLC01*. This POE is to be run later in a cyclical IRP task.
- Position one or several of the FBs CANop405\_PDO\_READ in this POE as needed.
- Connect the function blocks to variables of the correct data type and assign them the desired values. On the input us\_PDO\_NR, specify the number of the reception mailbox you have chosen. With the FB CANop405\_PDO\_INIT you have already made the assignment between COB\_ID and reception mailbox in the initialization. us\_DATALENGTH designates the valid bytes of the received PDO. It is the sum of all bytes of the mapped objects. The data of the received objects are output on d\_DATA0 and d\_DATA1. With the FB CANop405\_PDO\_READ, you also receive transmit PDOs, which you have requested via a remote request with the FB CANop405\_PDO\_WRITE.
- Create a cyclical bypass IRP task if such a task is not yet present in your project. With low time requirements for the PDO, you can also use a simple cyclical task with a priority of your choice. Integrate the created POE in this task.
- Translate the project and load it as a (boot) project on the PLC. Start the PLC.
- If necessary, carry out the configuration of the network nodes and set these to the OPERATIONAL state.

The PDO can be received with x\_ENABLE = 1. The FB CANop405\_PDO\_READ signals an arrived PDO with x\_CONFIRM = 1 and w\_ERROR = 16#0000. x\_CONFIRM is set to zero again in the next cycle and the next PDO can be received with x\_ENABLE = 1.

#### e) Example: Receiving PDO

We continue the example. The first transmit PDO of the BM4-O-CAN-03 option module is configured and we can move on to the actual data exchange via PDOs. First the network node must be switched into the OPERATIONAL state. For this purpose, we use the FB CANop405\_NMT again:

(\* Network control by FB CANop405\_NMT \*)

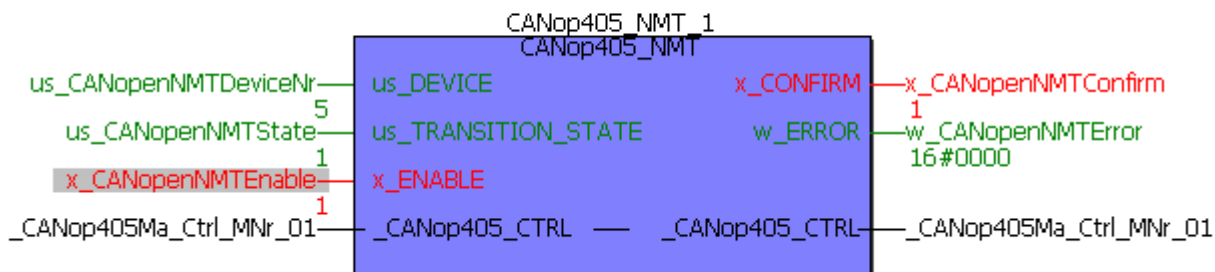


Figure 106: Switching CANopen slave into OPERATIONAL state

We position the FB CANop405\_PDO\_READ in the already existent POE with the FB CANop405\_PDO\_WRITE. As a result, the FB CANop405\_PDO\_READ is also called every 2 ms. As specified previously, the first reception mailbox is to be used, i.e.

## 4.4 Programming data exchange with PROPROG wt II and CANop405\_PLC01\_20bd03 library

us\_PDO\_NR is = 1. Due to the mapping, P301 lies in the first word of d\_DATA0 and P1040 in the second word of d\_DATA0, as well as in the first word of d\_DATA1. Translate the project and transmit it to the PLC. Start the project.

In the CANopen slave (b maXX controller with BM4-O-CAN-03), the PDO configuration (set and saved above) is active and the CANopen slave is in the OPERATIONAL state.

If you have switched off the CANopen slave, the stored configuration will be loaded when it is switched on again, i.e. the PDO configuration (saved above) will be used. However, the CANopen slave is in the PRE-OPERATIONAL state after being switched on, which means the CANopen slave must be switched into the OPERATIONAL state by the CANopen master via the network management (see [▶Example: Network management with NMT](#) on Page 108).

So that recognizable data traffic occurs, we also start the already existent FB CANop405\_PDO\_WRITE, which changes P1040. We can affect the status word via WinBASS II. The PDO reception can be started with x\_ENABLE = 1 on the FB CANop405\_PDO\_READ. PDOs should arrive immediately:

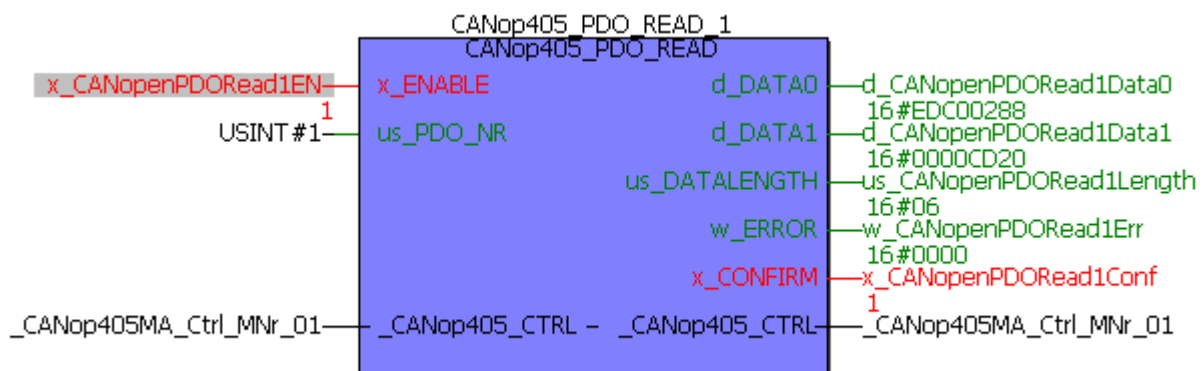


Figure 107: Reading data with FB CANop405\_PDO\_READ - Online

Please note that these are "snapshots" in the online mode and may differ from your values. The output x\_CONFIRM will also not continually have the value 1, as the PDOs are transmitted at a 5 ms time interval, however the FB CANop405\_PDO\_READ is called at a 2 ms time interval.

### 4.4.10 Synchronizing data exchange

#### 4.4.10.1 Definition according to CANopen specification

With many applications it is practical to synchronize the application of the input data and the transmission of the output data. CANopen provides the SYNC telegram for this purpose. The SYNC telegram has the highest priority after the NMT telegram and no user data. The SYNC telegram is transmitted according to the consumer/producer communication model, i.e. a network node generates a SYNC telegram, which is processed by one or several other network nodes. There is no confirmation of the receipt of the telegram. The reception serves the network nodes as a trigger for the reading of the received PDOs or for the transmission of PDOs, provided a PDO is configured for this transmission type

(see the chapter [▶4.4.9.2 Configuration of transmission properties of a PDO](#) from page 115 onward). The setting of whether a network node is a consumer or a producer is made with the Object 1005h:

SYNC telegram:

Object	Meaning
1005h	Configuration of SYNC telegram.
Subindex 0	COB-ID and validity

The value of Subindex 0 has the data type Unsigned32 and is divided up as follows:

Bit	Value/Meaning
31 (MSB)	<i>Not relevant</i>
30	0: Network node does not generate any SYNC telegrams 1: Network node generates SYNC telegrams
29	0: 11-bit CAN ID 1: 29-bit CAN ID (is not supported by the CANopen master module)
28 - 11	Is not supported by the CANopen master module
10 - 0 (LSB)	COB-ID of the SYNC telegram. As the CANopen master module uses the COB-ID according to the predefined identifier assignment (see the chapter <a href="#">▶4.2.2.3 Data exchange and objects of physical bus system</a> from page 53 onward), it must be 80h.

The default setting is almost always: SYNC consumer, i.e. you do not need to set the network node to SYNC consumer manually.

The CANopen master mode can be used as a SYNC producer. As the CANopen master module does not have any objects, we only need to use the function block CANop405\_SYNC to generate a SYNC telegram. The SYNC consumer function is not supported.

The application or transmission of the PDO contents runs on a SYNC consumer according to the following pattern:

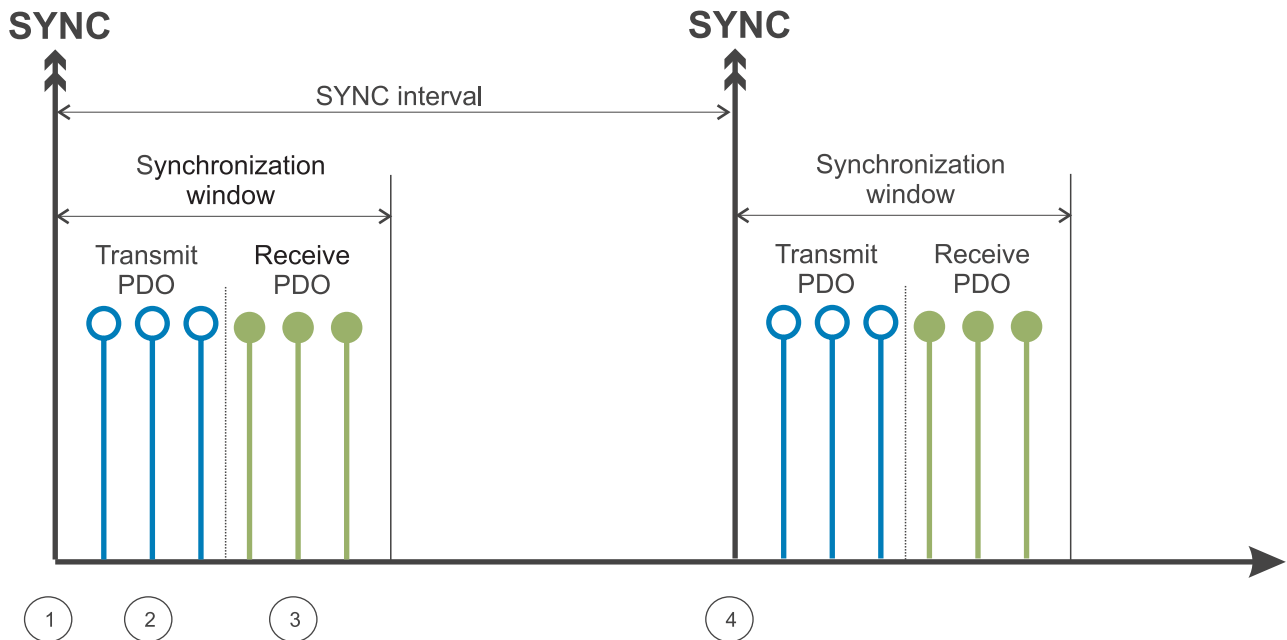


Figure 108: Synchronization mechanism

After the SYNC telegram is received, ① a network node first transmits the transmit PDOs (TxPDOs) ②. Then the PDOs received up to that point (RxPDOs) ③ are applied in a temporary memory. The RxPDOs are not processed until after the next SYNC telegram is received ④. Synchronous PDOs may only occur within the synchronization window. Some network nodes offer the option of monitoring the SYNC interval and can react accordingly if errors occur. There are also network nodes which not only synchronize the data transmission, but also internal processes to the SYNC telegram. In both cases, the SYNC interval in the Object 1006h must be set on the network node:

SYNC interval:

Object	Meaning
1006h	SYNC interval/duration of communication cycle.
Subindex 0	0: Not in use n: Length of SYNC interval in $\mu$ s.

Please note that some devices do not accept just any values of the object 1006h, but instead only certain values (e.g. 1000, 2000, 4000 etc.).

With the transmission type of the PDO (see the chapter [Configuration of transmission properties of a PDO](#) on Page 115), it can also be configured that a PDO is not applied until after the nth SYNC telegram. As a result, short cycles can be set up for important data and long cycles for less important data.

#### 4.4.10.2 Transmitting SYNC telegrams

---

SYNC telegrams are output by the CANopen master module with the FB CANop405\_SYNC. To use this function block, please proceed as follows:

- The FB CANop405\_SYNC should be implemented in a POE, which is called in the same time interval as the function blocks of the type CANop405\_PDO\_WRITE or CANop405\_PDO\_READ, and which transmit or receive synchronized PDOS. If a POE of this kind is not present, create it with PLC type SH03-30 and processor type *BMC\_M\_PLC01*. This POE is to be run later in a time event task.
- Connect the function block to variables of the correct data type. The SYNC telegram generation is started with `x_ENABLE = 1`.
- Create a cyclical bypass IRP task if such a task is not yet present in your project. Integrate the created POE in this task with the FB CANop405\_SYNC.
- Translate the project and load it as a (boot) project on the PLC.
- Start the project.

The SYNC telegram is output with `x_ENABLE = 1`. The FB CANop405\_SYNC signals a successful execution with `x_CONFIRM = 1` and `w_ERROR = 16#0000`. The SYNC telegram is generated during each run-through of the function block with `x_ENABLE = 1`. Re-setting with `x_ENABLE = 0` is not necessary.

#### 4.4.10.3 Example: Transmitting SYNC telegrams

---

We want to synchronize the data exchange via PDOS created in the chapters [▶4.4.9.6 Reading objects via PDOS](#) from page 132 onward and [▶4.4.9.5 Writing objects via PDOS](#) from page 120 onward with the BM4-O-CAN-03 option mode (on the b maXX controller from the device series of the b maXX drives). For this purpose the BM4-O-CAN-03 option module must first be in the PRE-OPERATIONAL state. Set the transmission type for the second receive PDO (Object 1401h) and the first transmit PDO (Object 1800h) to synchronous. The PDOS are to be applied or transmitted with every 10th SYNC telegram, i.e. the value 10 must be entered in Subindex 2 of the objects 1401h and 1800h. As soon as you change to the OPERATIONAL state again and release the transmission or reception of the PDOS, you will find that the parameters 1171 and 1040 in WinBASS II no longer change, although the FB CANop405\_PDO\_WRITE transmits data. The FB CANop405\_PDO\_READ also no longer outputs `x_CONFIRM = 1`. This is correct, as after all no synchronization has taken place yet, for the FB CANop405\_SYNC is still not implemented and active.

However first a few value still have to be configured. For this purpose the BM4-O-CAN-03 option module must first be in the PRE-OPERATIONAL state. First the source for the sync signal must be set on the b maXX controller with the parameter 531. We require "Use Sync 1 signal from the BACI". For this the value 2 (data length 1 byte) must be written to the object 4213h (P531), Subindex 0:

## 4.4 Programming data exchange with PROPROG wt II and CANop405\_PLC01\_20bd03 library

(\* Write object via SDO to device "us\_DeviceNumber" \*)

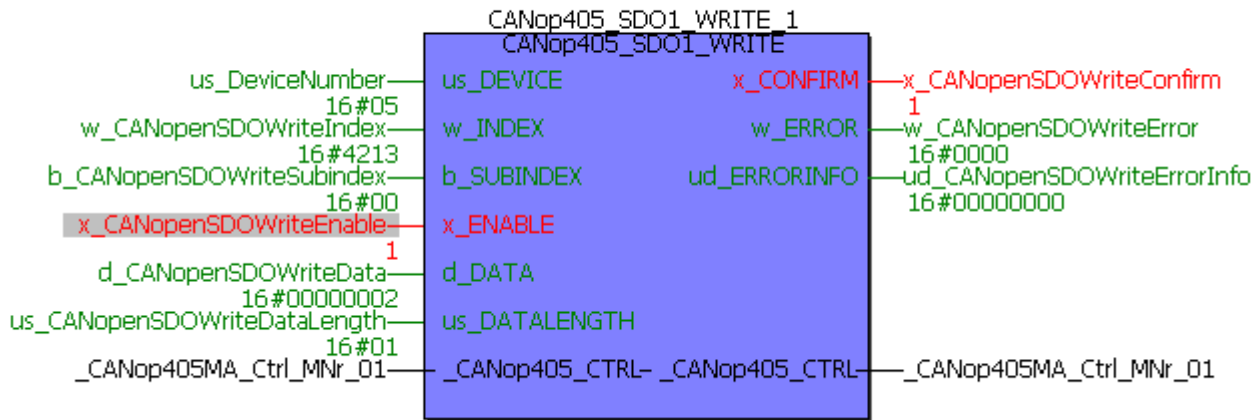


Figure 109: Source for sync signal on b maXX controller

The FB CANop405\_SYNC is to be called every 2 ms (= 2000  $\mu$ s = 7D0h  $\mu$ s). The object 1006h (duration of communication cycle) must also be set to this value:

(\* Write object via SDO to device "us\_DeviceNumber" \*)

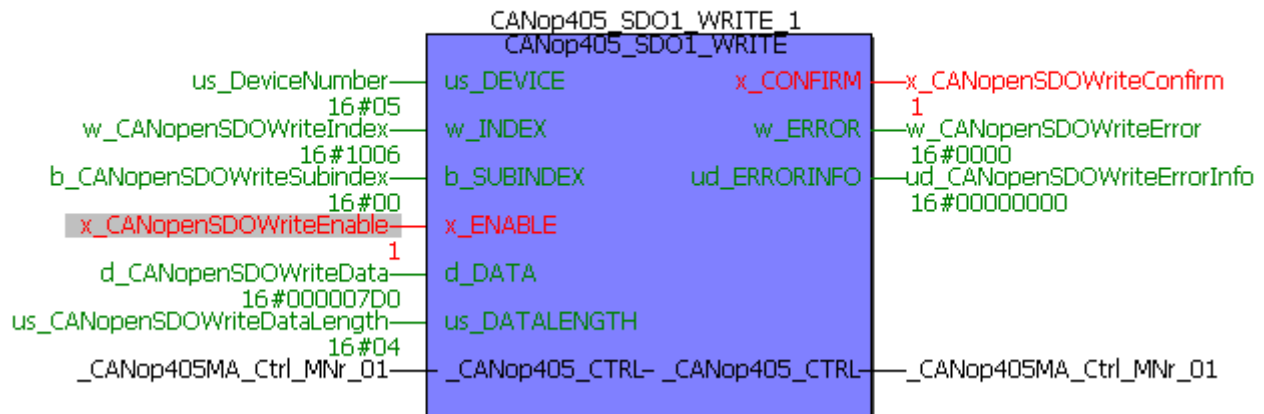


Figure 110: Setting communication cycle

This value is automatically applied to the parameter 532 "Sync Intervall". Save the data record of the b maXX controller via WinBASS II to avoid losing the set configuration when switching off/on.

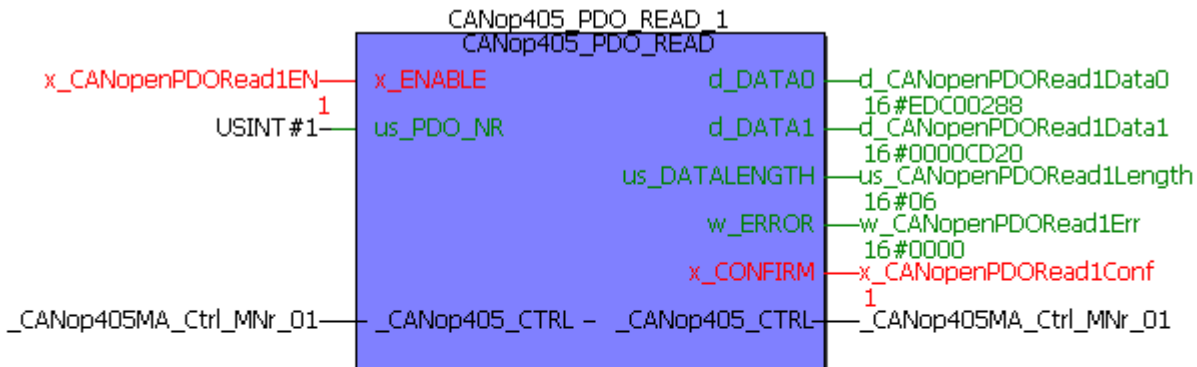
We implement the FB CANop405\_SYNC below the FB CANop405\_PDO\_READ. As a result, it is called automatically every 2 ms. Translate the project and transmit it to the PLC. Start the project.

In the CANopen slave (b maXX controller with BM4-O-CAN-03), the SYNC configuration (set above and saved with WinBASS II) is active and the CANopen slave is in the PRE-OPERATIONAL state.

If you have switched off the CANopen slave, the stored configuration will be loaded when it is switched on again, i.e. the SYNC configuration (saved above) will be used and the CANopen slave is in the PRE-OPERATIONAL state.



After the start, the BM4-O-CAN-03 option module must be switched into the OPERATIONAL state again. Release the FBs CANop405\_PDO\_READ and CANop405\_PDO\_WRITE. The FB CANop405\_PDO\_READ should still not receive and data. However, as soon as the FB CANop405\_SYNC is released, PDOs are received.



(\* Generation of the SYNC command \*)

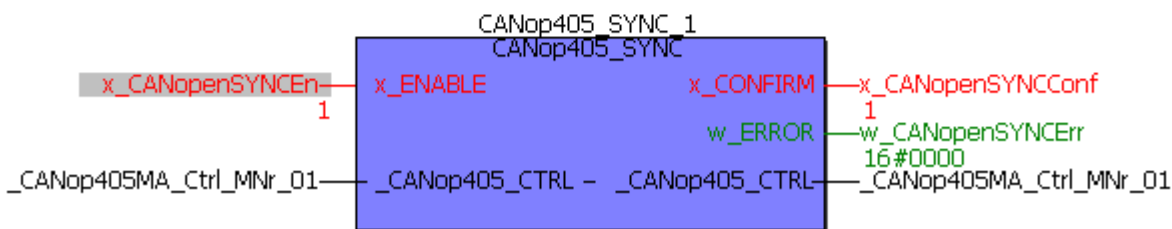


Figure 111: Generating SYNC telegram with FB CANop405\_SYNC

Please note that these are "snapshots" in the online mode and may differ from your values.

#### 4.4.11 Monitoring network nodes with Node Guarding

##### 4.4.11.1 Definition according to CANopen specification

For the failure monitoring of network nodes there is the Node Guarding mechanism and the Life Guarding mechanism. With Node Guarding the network nodes are monitored by the CANopen master, which can in turn detect the failure of the CANopen master via Life Guarding. Both mechanisms have the same functionality:

With Node Guarding the master requests a replay from the slave at certain time intervals 'Guard Time' via a remote telegram. The slave replies with the guarding message. This contains the node state (see the chapter [▶4.4.7 Starting individual network nodes - NMT](#) from page 106 onward) of the slaves and a toggle bit, which must change after every message. If the status or toggle bit do not match that expected by the master or if no response is received within a certain time period, i.e. the 'Life Time', then the master assumes a slave error. Conversely, the slave can detect the failure of the master. If the slave does not receive a message request from the master within the set 'Life Time', it assumes a master failure and sets, for example, its outputs to the error state and transmits an emergency telegram.

Two objects on the network node are important for setting the guarding functionality:

Guarding time:

Object	Meaning
100Ch	Configuration of Guarding Time.
Subindex 0	0: Guarding is not active n: Spacing between two guard telegrams in ms

Life time factor:

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

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 even with state changes, the toggle mechanism continues to be active. Node Guarding is started in the slave after the first guarding request telegram is received. From this point in time the monitoring time parameterized in the objects 100Ch and 100Dh runs in the slave.

#### 4.4.11.2 Node Guarding

---

The Node Guarding by the CANopen master module is carried out with the FB CANop405\_NODE\_GUARDING. To use this function block, please proceed as follows:

- The FB CANop405\_NODE\_GUARDING must be implemented in a POE with PLC type SH03-30 and processor type *BMC\_M\_PLC01*.
- Connect the function block to variables of the correct data type. Enter the number of the node to be monitored on `us_DEVICE`. `t_NODE_GUARD_TIME` is the Guarding Time and `us_LIFE_TIME_FACTOR` the Life Time Factor.
- Create a task in which you integrate the POE with the FB CANop405\_NODE\_GUARDING. The cycle time of this task is dependent on how quickly you want to detect a failure of a network node. The Node Guarding mechanism itself runs on the CANopen master module. This means it is not necessary to call the FB CANop405\_NODE\_GUARDING in the time interval of the Guard Time.
- Translate the project and load it as a (boot) project on the PLC.
- Configure the Guard Time (Object 100Ch) and the Life Time Factor (Object 100Dh) on the individual network nodes. To do this, use the function blocks for SDO communication.

Node Guarding is started with `x_ENABLE = 1`. The FB CANop405\_NODE\_GUARDING signals `x_NODE_OK = 1` as soon as the first valid response from the slave arrives. As long as further valid replies come from the slave, `x_NODE_OK` remains = 1. The state of the slave is output on `us_NODE_STATE`. The FB CANop405\_NODE\_GUARDING is the

only possibility to read out a node state. If an invalid response of the slave is received, or if no response is received within the Life Time, then an error message is output and `x_NODE_OK` becomes 0.

#### 4.4.11.3 Example: Monitoring a network node with Node Guarding

We want to monitor the BM4-O-CAN-03 option module (on the b maXX controller from the device series b maXX drives) for failure at a 500 ms interval with Node Guarding. To set the Node Guarding Time and the Life Time Factor, the BM4-O-CAN-03 option module must first be in the PRE-OPERATIONAL state. Now we set the object 100Ch to 500 ms:

```
(* Write object via SDO to device "us_DeviceNumber" *)
```



Figure 112: Setting Node Guarding Time on CANopen slave

For the Life Time Factor, Object 100Dh, we select the value 3:

```
(* Write object via SDO to device "us_DeviceNumber" *)
```

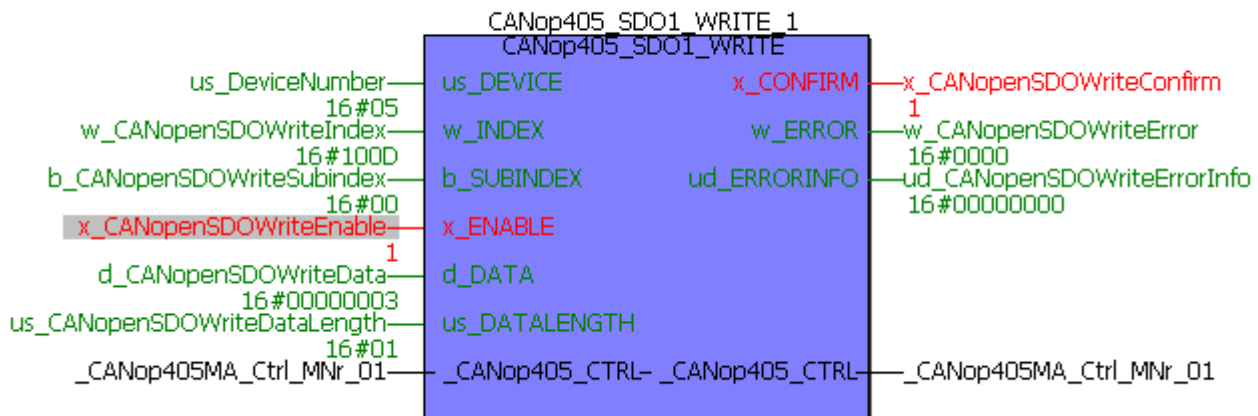


Figure 113: Setting Life Time Factor on CANopen slave

We implement the FB `CANop405_NODE_GUARDING` in the POE with the function block for the SDO communication. The time interval of the task assigned to this POE is easily

## 4.4 Programming data exchange with PROPROG wt II and CANop405\_PLC01\_20bd03 library

sufficient for our purposes to detect a node failure. After connecting the function block, translate the project and transmit it to the PLC. Start the project.

In the CANopen slave (b maXX controller with BM4-O-CAN-03), the Node Guarding configuration (set above) is active.

If you had switched off the CANopen slave, the BM4-O-CAN-03 option module will set the Guard Time and the Life Time Factor to the default values when it is switched on again. The Node Guarding configuration must be carried out again.

Node Guarding is possible in the OPERATIONAL and PRE-OPERATIONAL state. In the Online mode we can start the Node Guarding with `x_ENABLE = 1`.

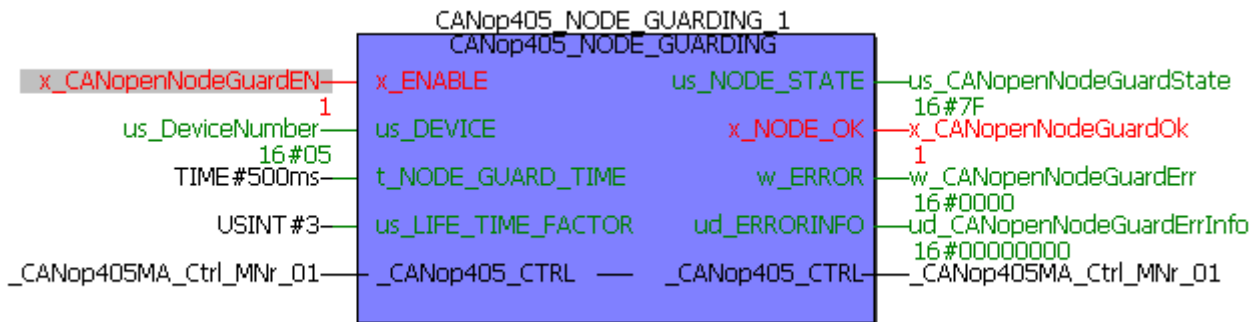


Figure 114: Node Guarding with FB CANop405\_NODE\_GUARDING

In this case the BM4-O-CAN-03 option module signals the PRE-OPERATIONAL state (7Fh). By disconnecting the CAN cable, we can generate the "Error Toggle Bit" error message:

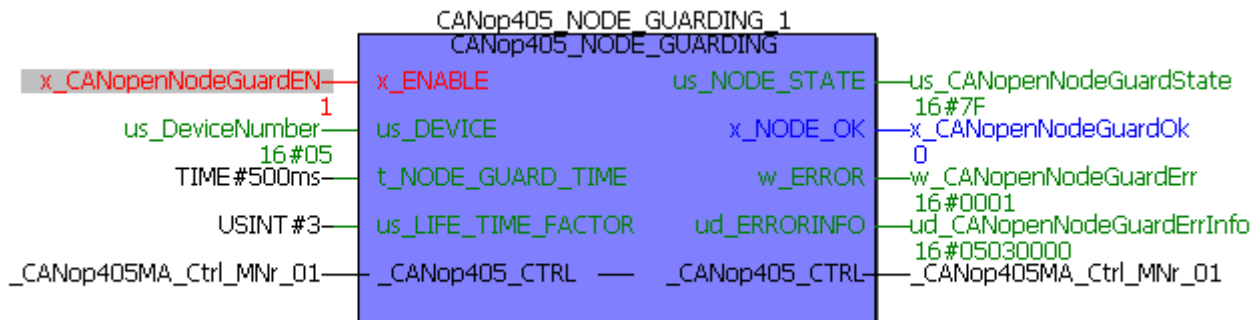


Figure 115: Node Guarding "Error Toggle Bit"

### 4.4.12 Receiving emergency telegrams

#### 4.4.12.1 Definition according to CANopen specification

Emergency telegrams are used by network nodes to signal errors. The emergency telegram is transmitted according to the consumer/producer communication model, i.e. a network node generates an emergency telegram, which is processed by one or several other network nodes. There is no confirmation of the receipt of the telegram. A new emergency

telegram is transmitted once for each newly added error. There is no repetition of the telegram. The CANopen master module can evaluate emergency telegrams from network nodes. It is not possible to transmit an emergency telegram.

The user data area of an emergency telegram is divided into three parts:

- 1 Emergency Error Code, 2 bytes
- 2 Error Register, 1 byte
- 3 Manufacturer-specific error code, 5 bytes

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
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

Error Code	Meaning
8210h	PDO not processed due to length error
8220h	PDO length exceeded
90xxh	External Error
F0xxh	Additional Functions
FFxxh	Device specific

xx = not defined

The object 1001h is contained in the Error Register. A network node can enter internal errors in the object 1001h. The coding according to CiA is as follows:

Bit	Meaning
0	Generic error
1	Current
2	Voltage
3	Temperature
4	Communication error
5	Device profile specific
6	Reserved
7	Manufacturer specific

For the further characterization of errors, 5 bytes are available for a manufacturer-specific error code. The coding of errors is highly device-specific due to the broad range of possibilities, and it is necessary to read the respective device documentation for further information. Please also note that most entries in the Emergency Error Code and Error Register are not obligatory. Also check which entries are supported in the device documentation.

#### 4.4.12.2 Evaluating emergency telegrams

Emergency telegrams are evaluated by the CANopen master module with the FB CANop405\_EMERGENCY. To use this function block, please proceed as follows:

- The FB CANop405\_EMERGENCY must be implemented in a POE with PLC type SH03-30 and processor type *BMC\_M\_PLC01*.
- Connect the function block to variables of the correct data type. On us\_DEVICE specify the number of the node to be monitored. If an emergency telegram is received, the received Emergency Error Code is output on w\_EMCY\_ERROR\_CODE, the Error Register on b\_ERROR\_REGISTER and the manufacturer-specific error info in the array a\_ERROR\_FIELD. If the received Error Code  $\gg 0$ , the output x\_EMERGENCY is set to 1. The outputs are reset with x\_RESET = 1.

- Create a task in which you integrate the POE with the FB CANop405\_EMERGENCY. The cycle time of this task is dependent on how quickly you want to detect a received emergency telegram.
- Translate the project and load it as a (boot) project on the PLC.

A received emergency telegram is displayed with `x_EMERGENCY = 1`. The function block need not be enabled separately; it is automatically active.

#### 4.4.12.3 Example: Receiving emergency telegram

We want to cause the BM4-O-CAN-03 option module (on the b maXX controller from the device series of the b maXX drives) to transmit an emergency telegram and then to evaluate this. A simple way to do this is simply to deactivate an active Node Guarding. Following this the BM4-O-CAN-03 option module must generate a Life Guarding error. According to the CiA definition (DS 301) and the documentation of the BM4-O-CAN-03 option module, a Life Guarding error delivers the following values:

<i>Emergency Error Code:</i>	8130h
<i>Error Register:</i>	At least Bit 0 and 4 set
<i>Manufacturer Code:</i>	0h, as not and error of the b maXX device

We implement the FB CANop405\_EMERGENCY in the POE with the function block for the SDO communication. After connecting the function block, translate the project and transmit it as a boot project to the PLC. Restart the b maXX 4400 device. Activate the Node Guarding as described in the chapter "Node Guarding". Do not forget to set the objects 100Ch (Guard Time) and 100Dh (Life Time Factor) via SDO. Activate the Node Guarding. The FB CANop405\_EMERGENCY may display an earlier emergency telegram. In this case, carry out a reset of the function block by briefly setting `x_RESET` to 1. Now switch off the Node Guarding on the FB CANop405\_NODE\_GUARDING with `x_ENABLE = 0`. Immediately following this, the FB CANop405\_EMERGENCY should display the receipt of the emergency telegram:

(\* check for received emergency messages of device nr. 5 \*)

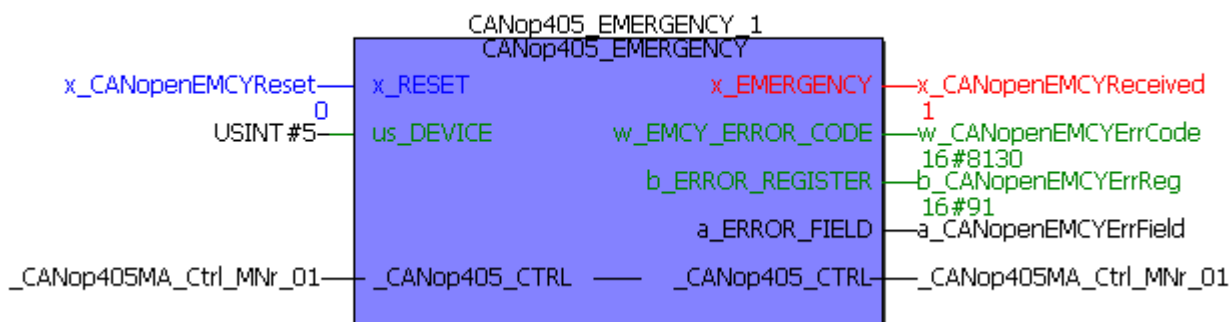


Figure 116: Evaluating emergency telegrams with FB CANop405\_EMERGENCY

## 4.4 Programming data exchange with PROPROG wt II and CANop405\_PLC01\_20bd03 library

Variable	Value	Default value	Type
[-] a_CANopenEMCYErrField			BYTE_8_BMARRAY
[0]	16#00		BYTE
[1]	16#00		BYTE
[2]	16#00		BYTE
[3]	16#00		BYTE
[4]	16#00		BYTE
[5]	16#00		BYTE
[6]	16#00		BYTE
[7]	16#00		BYTE

Figure 117: a\_ERROR\_FIELD in watch window of PROPROG wt II

We view the value of a\_ERROR\_FIELD in the watch window of PROPROG wt II. The displayed values match those we expected. The function block can be reset with x\_RESET = 1, enabling the next emergency telegram to be received:

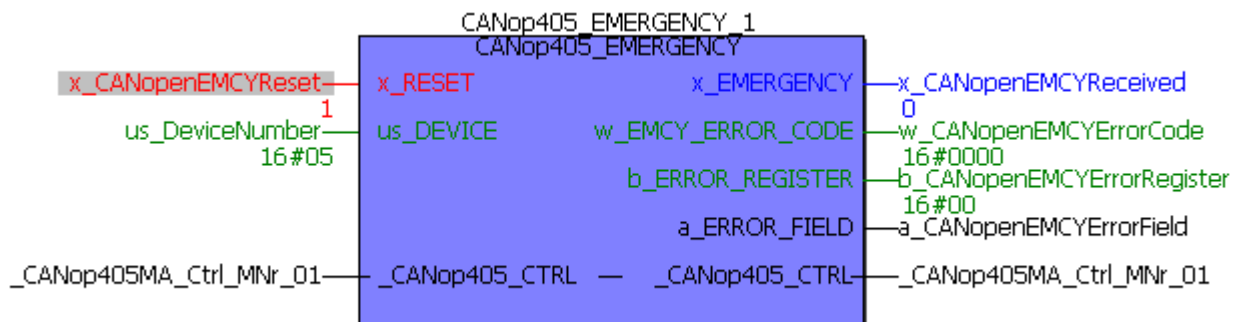


Figure 118: Resetting FB CANop405\_EMERGENCY



---

## 4.5 CANopen and Motion Control with PROProg wt II and motion configurator

---

### 4.5.1 General information on CANopen master module and Motion Control

---

Should you use the CANopen master module together with Motion Control, then a node in the CANopen network may not be operated at any time via the function blocks from the CANop405\_PLC01\_20bd03 library (or higher) together with Motion Control. This means a network node is either operated via the function blocks from the CANop405\_PLC01\_20bd03 library (or higher) or by Motion Control.

### 4.5.2 Initializing CANopen master module

---

When using Motion Control, the initialization of the CANopen master module is assumed by Motion Control. The function block CANop405\_INIT may no longer be used.

### 4.5.3 Data exchange via PDOs

---

When using the CANopen master module together with Motion Control, the number of PDOs to be written is reduced from 40 to the number of axes operated by Motion Control:

$$\text{PDOs to be written} \leq 40 - \text{number of Motion Control nodes}$$

As a result, the maximum transmission mailbox number to be assigned is also reduced to this value. The number of PDOs to be received is also decreased. The maximum number of PDOs to be received decreases from 63 by double the number of axes operated by:

$$\text{PDOs to be received} \leq 63 - 2 \times \text{number of Motion Control nodes}$$

As a result, the maximum reception mailbox number to be assigned is also reduced to this value.

The Motion Control configurator takes this reduction of the PDOs into account. However, observe these values when configuring your application.

### 4.5.4 Synchronizing data exchange via PDOs

---

When using Motion Control, the generation of the SYNC telegrams for the data exchange of is adopted by Motion Control. The function block CANop405\_SYNC may no longer be used. Of course, non-Motion Control nodes can be integrated in the synchronized data exchange through corresponding configuration.

### 4.6 Structure of CANopen telegrams

#### 4.6.1 General information

Due to the CANopen master module and the function blocks from the CANop405\_PLC01\_20bd03 library (and higher), the telegram traffic for you is completely encapsulated and you require only little knowledge of the structure of the individual CANopen telegrams. Should you nevertheless want to check the data traffic with a CAN analysis tool, the structure of the individual telegrams is contained in the following chapters. The telegrams are explained to you in the format

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

This structure is contained in most CAN analysis tools, whereby the data range Byte 0 to Byte 7 may still be adapted to the respective CANopen telegram. RTR refers to the bit for a Remote Transmit Request. The field DLC consists of 4 bits and refers to the number of data bytes (DLC = Data Length Code).

#### 4.6.2 NMT telegram

Two data bytes are transmitted per telegram. The data byte 0 contains the Command Specifier CS, and the data byte 1 the device address. If the address 0 is entered, then all nodes will be addressed with the corresponding command (broadcast).

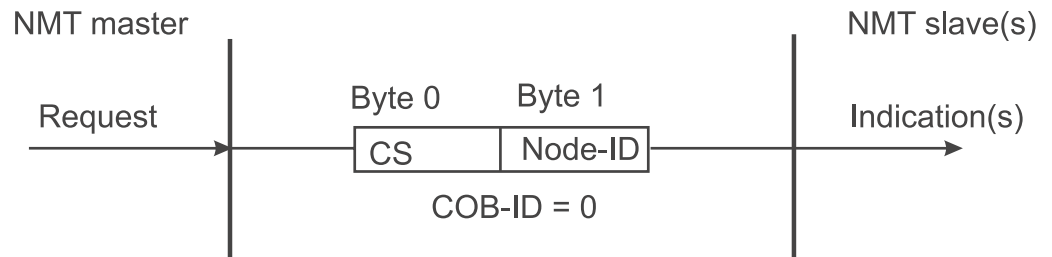


Figure 119: NMT communication relationship

CS	Designation	Effect
1	Start_Remote_Node	Starting of normal operation
2	Stop_Remote_Node	Deactivation of PDO and SDO communication
128	Enter_Pre-Operational_State	Transition to configuration mode
129	Reset_Node	Controlled resetting of entire object directory to default values
130	Reset_Communication	Resetting of communication section in object directory to default values

A telegram which switches node 16 into the configuration mode has the following appearance:

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

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

### 4.6.3 SDO telegram

#### 4.6.3.1 Structure of an SDO telegram

The COB-ID of the request SDO (request from master) results from 600h + address, and with response SDOs (response of slave) of 580h + address. The data field of the CAN data telegram (8 bytes) for an SDO is divided into three parts, i.e. a Command Specifier CS (1 byte), a Multiplexor M (3 bytes) and the actual user data range 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 multiplexer M consists of the 16-bit index of an object and the related eight-bit wide subindex. With segmented telegrams (not supported by the CANopen master module), the user data range is expanded by the three bytes of the multiplexer, which enables 7 bytes of user data to be transmitted per telegram. The Command Specifier CS classifies the various SDO types.

#### 4.6.3.2 Types of SDO transfer

The CANopen master module supports the expedited transfer according to DS 301. Objects can be written or read if their data comprise a maximum of 4 bytes. Only two telegrams are required, i.e. a request and a response.

#### 4.6.3.3 Writing object via SDO

The CANopen master module transmits (client) a write request to the slave (server). This carries out the request and acknowledges this with the response.

## 4.6 Structure of CANopen telegrams

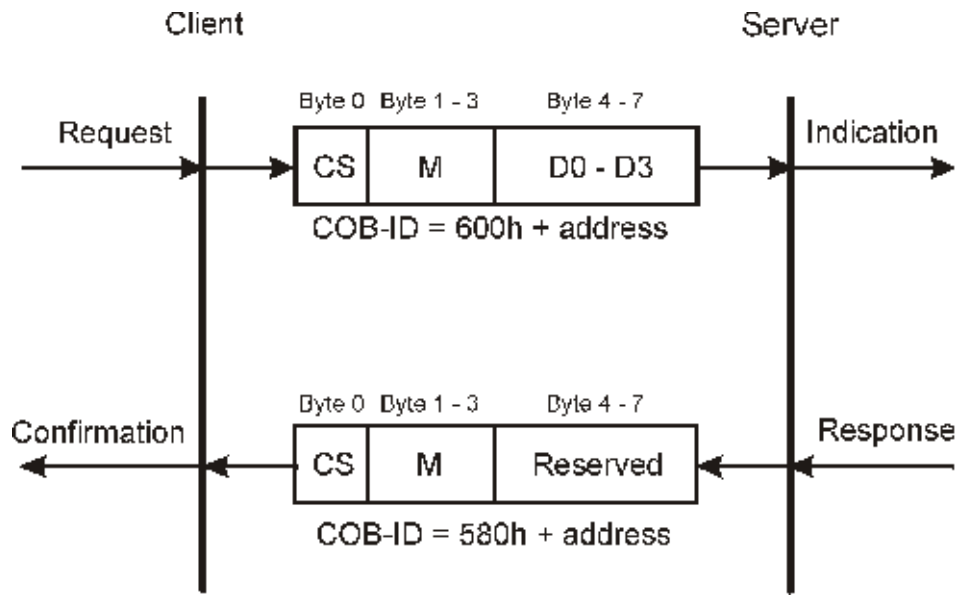


Figure 120: Writing SDO communication relationship

The Command Specifier CS for the request is dependent on the user data length. D0 is the LSB, D3 the MSB.

Data length in D0 - D3	Command Specifier CS
1 byte	2Fh
2 bytes	2Bh
4 bytes	23h

The Command Specifier CS for the response is 60h, the multiplexer M is identical to that of the request, the data field has no meaning (reserved).

Example:

The value "-3" (FDh) is to be written to the Object 6060h, Subindex 00h, of the slave with the address 4. The data width of this object is eight bits.

Request:

			CS	Multiplexor			D0	D1	D2	D3
COB-ID	RTR	DLC	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
604h	0	8h	2Fh	60h	60h	00h	FDh	00h	00h	00h

Response:

			CS	Multiplexor			D0	D1	D2	D3
COB-ID	RTR	DLC	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
584h	0	8h	2Fh	60h	60h	00h	00h	00h	00h	00h

The value "12" (0Ch) is to be written to the Object 43E9h, Subindex 00h, of the slave with the address 4. The data width of this object is 16 bits.

Request:

			CS	Multiplexor			D0	D1	D2	D3
COB-ID	RTR	DLC	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
604h	0	8h	2Bh	E9h	43h	00h	0Ch	00h	00h	00h

Response:

			CS	Multiplexor			D0	D1	D2	D3
COB-ID	RTR	DLC	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
584h	0	8h	60h	E9h	43h	00h	00h	00h	00h	00h

The value "60610008h" is to be written to the Object 1800h, Subindex 02h, of the slave with the address 4. The data width of this object is 32 bits.

Request:

			CS	Multiplexor			D0	D1	D2	D3
COB-ID	RTR	DLC	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
604h	0	8h	23h	00h	18h	02h	08h	00h	61h	60h

Response:

			CS	Multiplexor			D0	D1	D2	D3
COB-ID	RTR	DLC	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
584h	0	8h	60h	00h	18h	02h	00h	00h	00h	00h

#### 4.6.3.4 Reading object via SDO

The CANopen master module transmits (client) a read request to the slave (server). This carries out the request and acknowledges this with the response, which also contains the data.

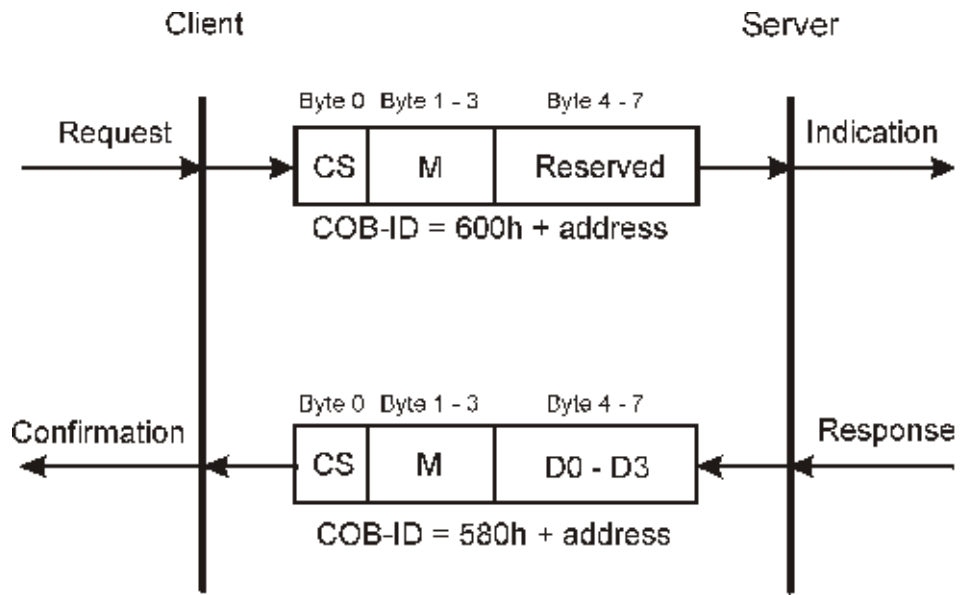


Figure 121: Reading SDO communication relationship

An SDO server (master) transmits a read request to the slave. This slave carries out the request and sends the requested data in the response telegram. The Command Specifier CS for the request is always 40h. The Command Specifier CS for the response is dependent on the user data length. D0 is the LSB, D3 the MSB.

Data length in D0 - D3	Command Specifier CS
1 bytes	4Fh
2 bytes	4Bh
4 bytes	43h

The multiplexer of the request and the response match.

Example:

The Object 6061h, Subindex 00h, of the slave with the address 4 is to be read. The data width of this object is 8 bits.

Request:

			CS	Multiplexor			D0	D1	D2	D3
COB-ID	RTR	DLC	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
604h	0	8h	40h	61h	60h	00h	00h	00h	00h	00h

Response:

			CS	Multiplexor			D0	D1	D2	D3
COB-ID	RTR	DLC	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
584h	0	8h	4Fh	61h	60h	00h	D0	00h	00h	00h

The Object 6041h, Subindex 00h, of the slave with the address 4 is to be read. The data width of this object is 16 bits.

Request:

			CS	Multiplexor			D0	D1	D2	D3
COB-ID	RTR	DLC	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
604h	0	8h	40h	41h	60h	00h	00h	00h	00h	00h

Response:

			CS	Multiplexor			D0	D1	D2	D3
COB-ID	RTR	DLC	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
584h	0	8h	4Bh	41h	60h	00h	D0	D1	00h	00h

The Object 1400h, Subindex 01h, of the slave with the address 4 is to be read. The data width of this object is 32 bits.

Request:

			CS	Multiplexor			D0	D1	D2	D3
COB-ID	RTR	DLC	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
604h	0	8h	40h	00h	14h	01h	00h	00h	00h	00h

Response:

			CS	Multiplexor			D0	D1	D2	D3
COB-ID	RTR	DLC	Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7
584h	0	8h	43h	00h	14h	01h	D0	D1	D2	D3

## 4.6 Structure of CANopen telegrams

### 4.6.4 PDO telegram

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

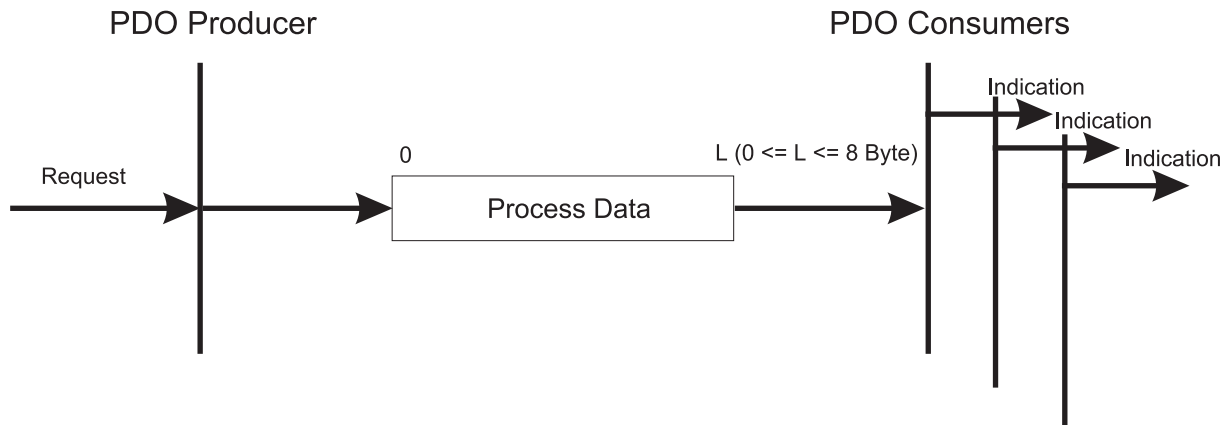


Figure 122: PDO communication relationship

All 8 bytes of the data range in the CAN data telegram are available for PDO telegrams. The content results from the settings of the mapping objects. For the COB-ID, values from 181h to 57Fh result, if the COB-ID was assigned in accordance with the predefined identifier assignment of the CiA. In addition, the COB-ID can also be set as desired.

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

With remote requests from PDO telegrams, the COB-ID of PDO request and PDO response has the same value. In the PDO request, the RTR bit (part of the DLC) is set. The data bytes have no contents, and the DLC field is therefore 0.

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



#### 4.6.5 SYNC telegram

A SYNC producer transmits the SYNC telegram to the CANopen network. Depending on the configuration, one or several SYNC consumers evaluate the telegram.

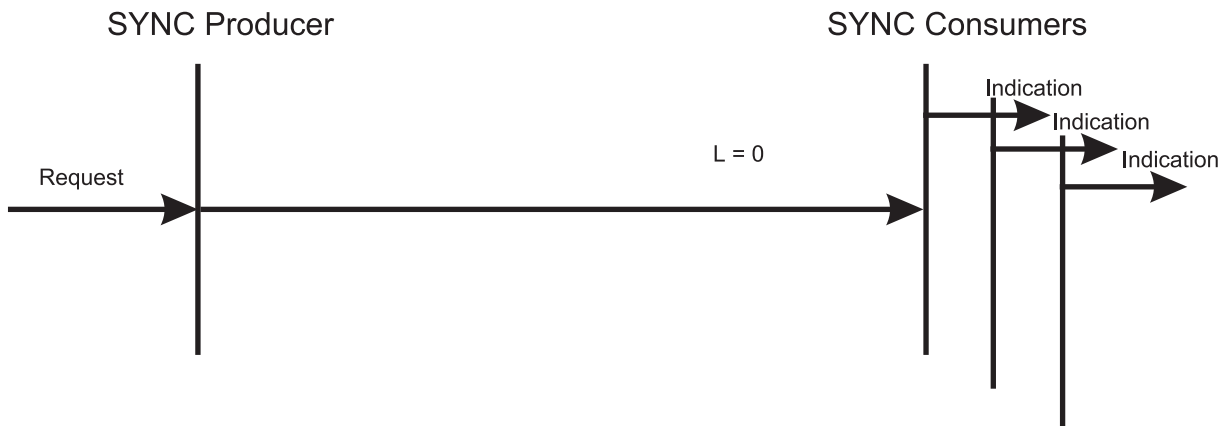


Figure 123: SYNC communication relationship

The SYNC telegram contains no data. The COB-ID is 80h according to the predefined identifier assignment of the CiA. In addition, the COB-ID can also be set as desired.

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

## 4.6 Structure of CANopen telegrams

### 4.6.6 Node Guarding telegram

The master interrogates the slaves at certain intervals with remote frames.

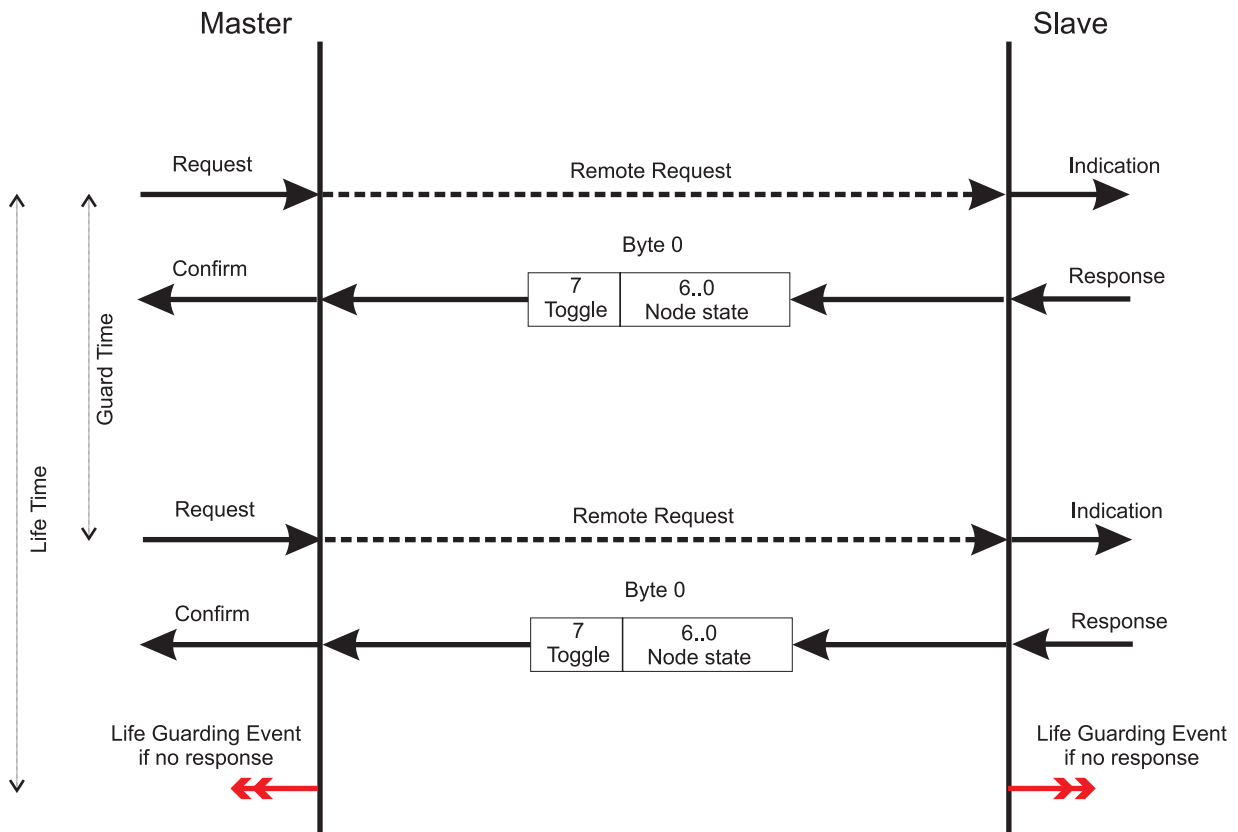


Figure 124: Node Guarding communication relationship

The COB-ID results from 700h + address. The data field of the CAN data telegram consists in the response telegram of one byte, i.e. the Byte 0. Byte 0 is divided into the Toggle Bit (Bit 7) and the Node State (Bit 6 - 0).

COB-ID	RTR	DLC	Byte 0
700h + address	0	1h	Toggle Bit + Node State

The remote request contains no data. The RTR bit is set.

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

#### 4.6.7 Emergency telegram

An emergency producer transmits the emergency telegram to the CANopen network. Depending on the configuration, one or several emergency consumers evaluate the telegram.

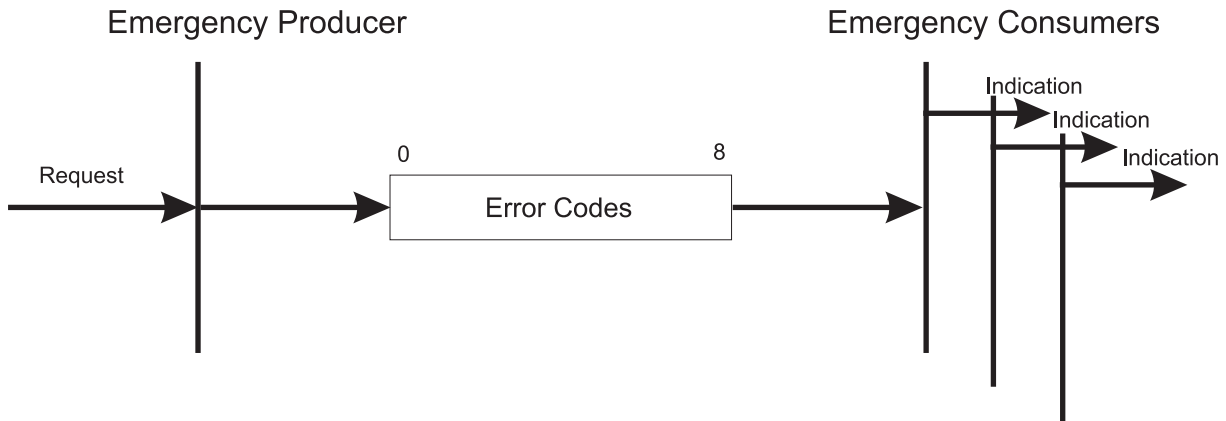


Figure 125: Emergency communication relationship

The COB-ID results from 80h + address. All 8 bytes of the CAN data telegram are available for emergency telegrams. The 8 bytes are divided up 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

<b>API</b>	Applications Programming Interface	<b>DS</b>	Draft Standard
<b>ARP</b>	Address Resolution Protocol	<b>DSP</b>	Draft Standard Proposal
<b>BACI</b>	Baumüller Component Interface	<b>EDS</b>	Electronic Data Sheet
<b>BM4-O-CAN-03</b>	CANopen-Slave option module at the b maXX controller from the device line b maXX drives	<b>EMC</b>	Electromagnetic compatibility
<b>BUB</b>	Ballast unit	<b>EN</b>	European standard
<b>BUC</b>	Baumüller feed/return feed unit	<b>EPROM</b>	Erasable Programmable Read Only Memory
<b>BUG</b>	Baumüller converter basic feed unit	<b>ESD</b>	Electrostatic Sensitive Device
<b>BUM</b>	Baumüller individual power unit	<b>FTP</b>	File Transfer Protocol
<b>BUS</b>	Baumüller power module	<b>HD</b>	Hamming distance
<b>CAL</b>	CAN Application Layer	<b>HTML</b>	Hypertext Markup Language
<b>CAN</b>	Controller Area Network	<b>HTTP</b>	Hypertext Transfer Protocol
<b>CiA</b>	CAN in Automation	<b>I/O</b>	Input/Output
<b>COB</b>	Communication Object	<b>ICMP</b>	Internet Control Message Protocol
<b>COB-ID</b>	Communication Object Identifier	<b>IP</b>	Internet Protocol
<b>CSMA/CD</b>	Carrier Sense Multiple Access / Collision Detection	<b>IRP</b>	Interrupt
<b>CSMA/CA</b>	Carrier Sense Multiple Access / Collision Avoidance	<b>ISO</b>	International Organization for Standardization
<b>CPU</b>	Central Processing Unit	<b>LAN</b>	Local Area Network
<b>DC</b>	d.c. current	<b>LED</b>	Light Emitting Diode
<b>DCF</b>	Device Configuration File	<b>LSS</b>	Layer Setting Services
<b>DHCP</b>	Dynamic Host Configuration Protocol	<b>MAC</b>	Media Access Control
<b>DIN</b>	Deutsches Institut für Normung e.V. (German Standards Institute)	<b>OSI</b>	Open Systems Interconnect
<b>DP-RAM</b>	Dual-port RAM	<b>PDD</b>	Process Data Directory
<b>DR</b>	Draft Recommendation	<b>PDO</b>	Process Data Object
		<b>PLC</b>	Process Loop Controller (PLC)
		<b>RAM</b>	Random Access Memory
		<b>SAP</b>	Service Access Point
		<b>SDO</b>	Service Data Object
		<b>SMS</b>	Short Message System
		<b>SMTP</b>	Simple Mail Transfer Protocol

<b>PLC</b>	Programmable Logic Controller
<b>SRD</b>	SDO Requesting Device
<b>SRDO</b>	Safety Relevant Data Object
<b>TCP</b>	Transport Control Protocol
<b>Telnet</b>	Terminal over Network
<b>UDP</b>	User Datagram Protocol
<b>URL</b>	Uniform Resource Locator
<b>USS</b>	USS protocol function module
<b>USS<sup>®</sup></b>	Trademark of Siemens, universal serial interface
<b>VDE</b>	Verband deutscher Elektrotechniker (German Association of Electrical Engineers)
<b>WWW</b>	World Wide Web
<b>16#</b>	Prefix for hexadecimal numbers



## APPENDIX B - TABLES

### B.1 CANopen objects pursuant to DS 301

---

The following is a list of frequently used objects pursuant to the CANopen communication profile DS 301. Note that the CANopen master module with software version < **01.20** (i.e. BMC-M-ETH-02/CAN-04-01-00-001-**000**) itself does not contain these object.

In conjunction with ProMaster, the CANopen master module has its own CANopen objects. An overview of the objects of the CANopen master module with software version  $\geq$  **01.20** (i.e.  $\geq$  BMC-M-ETH-02/CAN-04-01-00-001-**001**) is found in [►B.3 CANopen objects of the CANopen masters \(software version  \$\geq\$  01.20\)](#) from Page 182.

Note that the CANopen master module with software version < **01.20** itself does not contain any objects.

Overview of the objects:

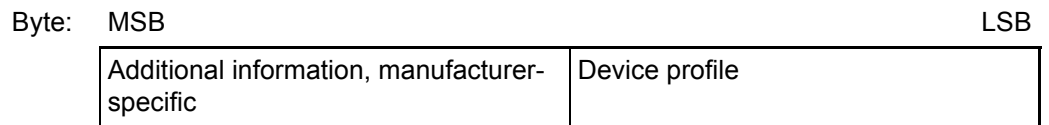
Index	Object contents/Meaning
<b>1000h</b>	Device type: supported device profile and additional information
<b>1001h</b>	Error register
<b>1003h</b>	Error memory contains a device error list
<b>1005h</b>	SYNC telegram COB-ID
<b>1006h</b>	SYNC interval
<b>100Ch</b>	Guarding Time
<b>100Dh</b>	Life Time Factor
<b>1010h</b>	Save parameters
<b>1011h</b>	Load default parameters
<b>1400h - 15FFh</b>	Reception PDO (RxPDO) communication parameters
<b>1600h - 17FFh</b>	Reception PDO (RxPDO) mapping parameters
<b>1800h - 19FFh</b>	Transmission PDO (TxPDO) communication parameters
<b>1A00h - 1BFFh</b>	Transmission PDO (TxPDO) mapping parameters

### 1000h - device type

Specifies the supported device profile and additional information on the device

Index	Subindex	Name	Data type	Access	Category	Map-able
1000h	0h	Device Type	Unsigned32	ro	Mandatory	No

The 32-bit value is divided into two 16-bit values:



### 1001h - error register

Contains a bit list with active device errors

Index	Subindex	Name	Data type	Access	Category	Map-able
1001h	0h	Error Register	Unsigned8	ro	Mandatory	Optional

The individual bits of the 8-bit value have the following meaning:

Bit	Meaning	Category
0	Generic error	Mandatory
1	Current	Optional
2	Voltage	Optional
3	Temperature	Optional
4	Communication error	Optional
5	Device profile specific	Optional
6	Reserved	Optional
7	Manufacturer specific	Optional

If a bit is set, the error has occurred.

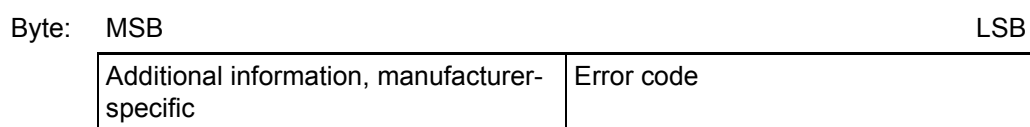
### 1003h - Error memory

Contains an error list of the device starting with the newest.



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

Subindex 0h contains the number of entered errors. The error list can be deleted by writing the value 0 to subindex 0h. Subindex 1h contains the current error. If a new error occurs, it is entered at subindex 1h and all others move down one subindex. The 32-bit value of the standard error field is divided into two 16-bit values:



The error code is yielded after the *Emergency Error Code* is defined in the emergency telegram:

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

Error code	Meaning
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

### 1005h - SYNC telegram COB-ID

Contains the COB-ID of the SYNC telegram and specifies whether the device generates the SYNC telegram.

Index	Subindex	Name	Data type	Access	Category	Map-able
1005h	0h	COB-ID SYNC	Unsigned32	rw	Conditional	No

The object is in the Mandatory category if synchronous PDO transmission is supported.

The individual bits of the 32-bit value have the following meaning:

Bit	Value/Meaning
31 (MSB)	Not relevant
30	0: Network node does not generate SYNC telegram 1: Network node generates SYNC telegram
29	0: 11-bit CAN ID 1: 29-bit CAN ID (not supported by CANopen master module)
28 - 11	Bits 28 - 11 with 29-bit CAN ID (not supported by CANopen master module)
10 - 0 (LSB)	COB-ID of the SYNC telegram. Since the CANopen master module uses the COB-ID after predefined issuing of the identifier, it must be 80h.

**1006h - SYNC interval**

Length of the SYNC interval in  $\mu$ s.

Index	Subindex	Name	Data type	Access	Category	Map-able
1006h	0h	Communication cycle period	Unsigned32	rw	Conditional	No

The object is in the Mandatory category for network nodes that generate the SYNC telegram. In this case, the period is entered in  $\mu$ s here. Network nodes that evaluate the SYNC telegram can use this value for monitoring and to synchronize internal communication cycles.

**100Ch - Guarding Time**

Interval between two guarding telegrams in ms.

Index	Subindex	Name	Data type	Access	Category	Map-able
100Ch	0h	Guard time	Unsigned16	rw	Conditional	No

The object contains the interval between two guarding telegrams in ms. The object is in the Mandatory category for network nodes that do not support the heartbeat. Since monitoring of the network nodes via a heartbeat is not possible with the CANopen master module, the object must be present.

**100Dh - Life Time Factor**

Yields the life time in ms when multiplied by the guarding time (object 100Ch).

Index	Subindex	Name	Data type	Access	Category	Map-able
100Dh	0h	Life time factor	Unsigned8	rw	Conditional	No

The object is in the Mandatory category for network nodes that do not support the heartbeat. Since monitoring of the network nodes via a heartbeat is not possible with the CANopen master module, the object must be present.

**1010h - Save parameters**

Enables the saving of parameters in nonvolatile memory.

Index	Subindex	Name	Data type	Access	Category	Map-able
<b>1010h</b>	-	Store parameters	Array	-	Optional	-
	0h	Largest subindex supported	Unsigned8	ro	Mandatory	No
	1h	Save all parameters	Unsigned32	rw	Mandatory	No
	2h	Save communication parameters	Unsigned32	rw	Optional	No
	2h	Save application parameters	Unsigned32	rw	Optional	No
	4h - 7h	Save manufacturer defined	Unsigned32	rw	Optional	No

Writing the "save" command on the respective subindex saves a specific parameter selection. The "save" command is represented by the following 32-bit value:

Byte:   MSB LSB

e	v	a	s
65h	76h	61h	73h

The scope of the parameter selection is yielded from the name of the subindex.

Reading a subindex provides the following information:

Bit	Value/Meaning
31 - 2	<i>Reserved</i>
1	0: The device does not save the parameters automatically (e.g. when it is switched off) 1: The device saves the parameters automatically (e.g. when it is switched off)
0	0: Parameters cannot be saved by command 1: Parameters can be saved by command

### 1011h - Load default parameters

Enables activation of default parameters.

Index	Subindex	Name	Data type	Access	Category	Map-able
<b>1011h</b>	-	Restore default parameters	Array	-	Optional	-
	0h	Largest subindex supported	Unsigned8	ro	Mandatory	No
	1h	restore all parameters	Unsigned32	rw	Mandatory	No

Index	Subindex	Name	Data type	Access	Category	Map-able
	2h	restore communication parameters	Unsigned32	rw	Optional	No
	2h	restore application parameters	Unsigned32	rw	Optional	No
	4h - 7h	restore manufacturer defined	Unsigned32	rw	Optional	No

By writing the "load" command on the respective subindex, a specific default parameter selection is loaded and activated after the next device reset. The "load" command is represented by the following 32-bit value:

Byte:   MSBLSB

d	a	o	l
64h	61h	6Fh	6Ch

The scope of the parameter selection is yielded from the name of the subindex.

Reading a subindex provides the following information:

Bit	Value/Meaning
31 - 1	<i>Reserved</i>
0	0: Default parameters cannot be activated 1: Default parameters can be activated

#### 1400h-15FFh - Reception PDO (RxPDO) communication parameters

The objects 1400h to 15FFh determine the properties for transmission of RxPDO 1 to RxPDO 512. Object 1400h describes RxPDO 1 and, if supported by the device, object 15FFh describes the RxPDO 512.

Index	Subindex	Name	Data type	Access	Category	Map-able
<b>1400h - 15FFh</b>	-	RxPDO parameters	CiA defined	-	Conditional	-
	0h	Largest subindex supported	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** specifies the number of the last valid subindex. The individual bits of the 32-bit value of **subindex 1** have the following meaning:

Subindex 1, bit	Value/Meaning
31 (MSB)	0: PDO exists and is valid 1: PDO does not exist or is invalid After predefined issuing of the identifier, only four PDOs for writing and four PDOs for reading are possible. If a device supports more than these four PDOs, the COB-IDs for the additional PDOs themselves must be issued (bits 10 - 0 in this subindex). For this reason, bit 31 of the additional PDOs usually has a value of 1 and is only set to 0 by the user as soon as the user issues a COB-ID.
30	0: The PDO can be requested via a remote request 1: The PDO cannot be requested via a remote request
29	0: 11-bit CAN ID 1: 29-bit CAN ID (not supported by CANopen master module)
28 - 11	Bits 28 - 11 with 29-bit CAN ID (not supported by CANopen master module)
10 - 0 (LSB)	COB-ID of the PDO. The first four PDOs for reading have the COB-IDs after predefined issuing of the identifier with the default settings.

The transmission type of the PDO is specified with **subindex 2**:

Subindex 2, value	Type	Meaning
0	Synchronous	PDO with appropriate COB-ID received before the last SYNC telegram is accepted
1 - 240	Synchronous	PDO with appropriate COB-ID received before the last SYNC telegram is accepted
252	Remote	<i>Not possible</i>
253	Remote	<i>Not possible</i>
254	Asynchronous	Each PDO with an appropriate COB-ID is accepted upon reception
255	Asynchronous	Each PDO with an appropriate COB-ID is accepted upon reception

### 1600h-17FFh - Reception PDO (RxPDO) mapping parameters

The objects 1600h to 17FFh determine the mapped objects of RxPDO 1 to RxPDO 512. Object 1600h describes RxPDO 1 and, if supported by the device, object 17FFh describes the RxPDO 512.

Index	Subindex	Name	Data type	Access	Category	Map-able
<b>1600h - 17FFh</b>	-	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 specifies the number of mapped objects in the PDO. As soon as the mapping of a PDO is changed, the number of mapped objects must first be set to zero. The mapping can then be set, and the number of mapped objects in subindex 0 must finally be entered. The mapping information of the objects in the PDO are specified in subindices 1h to 40h. Subindex 1h contains the information for the first mapped object, subindex 2h for the second mapped object etc.. A maximum of 64 (40h) objects can be mapped, whereby the maximum data range of 8 bytes of the PDO may not be exceeded. The contents of the 32-bit value of subindices 1h - 40h have the following meaning:

Byte:   MSBLSB

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

The objects are mapped in a PDO one after another.

#### 1800h-19FFh - Transmission PDO (TxPDO) communication parameters

The objects 1800h to 19FFh determine the properties for transmission of TxPDO 1 to TxPDO 512. Object 1800h describes TxPDO 1 and, if supported by the device, object 19FFh describes the TxPDO 512.

Index	Subindex	Name	Data type	Access	Category	Map-able
<b>1800h - 19FFh</b>	-	TxPDO parameters	CiA defined	-	Conditional	-
	0h	Largest subindex 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** specifies the number of the last valid subindex. The individual bits of the 32-bit value of **subindex 1** have the following meaning:

Subindex 1, bit	Value/Meaning
31 (MSB)	0: PDO exists and is valid 1: PDO does not exist or is invalid After predefined issuing of the identifier, only four PDOs for writing and four PDOs for reading are possible. If a device supports more than these four PDOs, the COB-IDs for the additional PDOs themselves must be issued (bits 10 - 0 in this subindex). For this reason, bit 31 of the additional PDOs usually has a value of 1 and is only set to 0 by the user as soon as the user issues a COB-ID.
30	0: The PDO can be requested via a remote request 1: The PDO cannot be requested via a remote request
29	0: 11-bit CAN ID 1: 29-bit CAN ID (not supported by CANopen master module)
28 - 11	Bits 28 - 11 with 29-bit CAN ID (not supported by CANopen master module)
10 - 0 (LSB)	COB-ID of the PDO. The first four PDOs for writing have the COB-IDs after predefined issuing of the identifier with the default settings.

The transmission type of the PDO is specified with **subindex 2**:

Subindex 2, value	Type	Meaning
0	Synchronous	Transmission occurs after every SYNC telegram is received
1 - 240	Synchronous	Transmission occurs after reception of the set number (1 - 240) of SYNC telegrams
252	Remote	When a SYNC telegram is received, the PDO is updated. Transmission does not occur until a remote request is received.
253	Remote	Updating and transmission occur after a remote request is received.
254	Asynchronous	Transmission is manufacturer-specific. Note: Most devices transfer the PDO when a time event occurs (expiration of a timer) with this type. The time is set under subindex 5.
255	Asynchronous	Transmission depends on the supported device profile. Note: With this type, most devices transfer the PDO as soon as one of the mapped values has changed.



With **subindex 3**, a transmission delay of a multiple of 100 µs can be set for the PDO. It does not extend the response time with the first value change, but is active when subsequent changes are made immediately afterward. The inhibit time (transmission delay time) describes the span of time that must be allowed to pass between transmission of two identical telegrams.

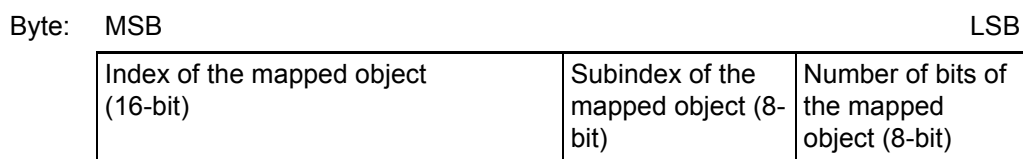
The time in ms for event-based PDO transfer is set in **subindex 5** if a device supports event-based PDO transmission (Subindex 2 = 254 or 255, depending on the device profile).

**1A00h-1BFFh - Transmission PDO (TxPDO) mapping parameters**

The objects 1A00h to 1BFFh determine the mapped objects of TxPDO 1 to TxPDO 512. Object 1A00h describes TxPDO 1 and, if supported by the device, object 1AFFh describes the TxPDO 512.

Index	Subindex	Name	Data type	Access	Category	Map-able
<b>1A00h - 1BFFh</b>	-	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** specifies the number of mapped objects in the PDO. As soon as the mapping of a PDO is changed, the number of mapped objects must first be set to zero. The mapping can then be set, and the number of mapped objects in subindex 0 must finally be entered. The mapping information of the objects in the PDO are specified in **subindices 1h to 40h**. Subindex 1h contains the information for the first mapped object, subindex 2h for the second mapped object etc.. A maximum of 64 (40h) objects can be mapped, whereby the maximum data range of 8 bytes of the PDO may not be exceeded. The contents of the 32-bit value of **subindices 1h to 40h** have the following meaning:



The objects are mapped in a PDO one after another.

### B.2 Default mapping of the PDO

Default mappings are defined in the device profiles. The following is an overview of the default mapping values for device profiles DS 401 and DSP 402. The assignment shown below is based on the maximum expansion of a device. Fewer objects can be mapped, depending on the hardware. This does not mean that objects of another type (e.g. analog inputs instead of digital inputs) move up in the mapping, however. If objects are dispensed with, the corresponding spots in the default mapping remain open. Naturally, you can assign these open spots with other objects as desired.

#### B.2.1 Default mapping for I/O modules pursuant to DS 401

Object	Meaning	Mapping value		
		Index	Subindex	Length
<b>1600h</b>	First reception 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	
<b>1601h</b>	Second reception 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	
<b>1602h</b>	Third reception 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	
<b>1603h</b>	Fourth reception PDO.			
Subindex 0	Number of mapped objects		4	
Subindex 1	Analog output 9 (16-bit resolution)	6411 09	10h	

Object	Meaning	Mapping value Index Subindex Length
Subindex 2	Analog output 10 (16-bit resolution)	6411 0A 10h
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
<b>1A00h</b>	First transmission 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
<b>1A01h</b>	Second transmission PDO.	
Subindex 0	Number of mapped objects	4
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
<b>1A02h</b>	Third transmission 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
<b>1A03h</b>	Fourth transmission 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 10h
Subindex 3	Analog input 11 (16-bit resolution)	6401 0B 10h
Subindex 4	Analog input 12 (16-bit resolution)	6401 0C 10h

## B.2 Default mapping of the PDO

### B.2.2 Default mapping for drives pursuant to DSP 402

Object	Meaning	Mapping value		
		Index	Subindex	Length
<b>1600h</b>	First reception PDO.			
Subindex 0	Number of mapped objects		1	
Subindex 1	Control word (16-bit)	6040 00	10h	
<b>1601h</b>	Second reception 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	
<b>1602h</b>	Third reception 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	20h	
<b>1603h</b>	Fourth reception PDO.			
Subindex 0	Number of mapped objects		2	
Subindex 1	Control word (16-bit)	6040 00	10h	
Subindex 2	Setpoint speed value (32-bit)	60FF 00	20h	
<b>1604h</b>	Fifth reception 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	
<b>1605h</b>	Sixth reception PDO.			
Subindex 0	Number of mapped objects		2	
Subindex 1	Control word (16-bit)	6040 00	10h	
Subindex 2	Setpoint speed value (16-bit)	6042 00	10h	
<b>1606h</b>	Seventh reception 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	
<b>1607h</b>	Eighth reception 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
<b>1A00h</b>	First transmission PDO.			
Subindex 0	Number of mapped objects		1	
Subindex 1	Status word (16-bit)	6041 00	10h	
<b>1A01h</b>	Second transmission PDO.			
Subindex 0	Number of mapped objects		2	
Subindex 1	Status word (16-bit)	6041 00	10h	
Subindex 2	Display operating mode (8-bit)	6061 00	08h	
<b>1A02h</b>	Third transmission PDO.			
Subindex 0	Number of mapped objects		2	
Subindex 1	Status word (16-bit)	6041 00	10h	
Subindex 2	Actual position value (32-bit)	6064 00	20h	
<b>1A03h</b>	Fourth transmission PDO.			
Subindex 0	Number of mapped objects		2	
Subindex 1	Status word (16-bit)	6041 00	10h	
Subindex 2	Actual speed value (32-bit)	606C 00	20h	
<b>1A04h</b>	Fifth transmission PDO.			
Subindex 0	Number of mapped objects		2	
Subindex 1	Status word (16-bit)	6041 00	10h	
Subindex 2	Actual torque value (32-bit)	6077 00	20h	
<b>1A05h</b>	Sixth transmission PDO.			
Subindex 0	Number of mapped objects		2	
Subindex 1	Status word (16-bit)	6041 00	10h	
Subindex 2	Actual speed value (16-bit)	6044 00	10h	
<b>1A06h</b>	Seventh transmission 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.3 CANopen objects of the CANopen masters (software version $\geq$ 01.20)

#### B.3.1 Overview of the objects

Index	Object contents/Meaning
1000h	Device type: supported device profile and additional information
1001h	Error register
1003h	Error memory contains a device error list
1005h	SYNC telegram COB-ID
1006h	SYNC interval
1008h	Manufacturer device name
1009h	Manufacturer hardware version
100Ah	Manufacturer software version
1010h	Save parameters
1011h	Load default parameters
1014h	EMCY telegram COB-ID
1016h	Heartbeat times of network nodes
1017h	Heartbeat time of CANopen master
1018h	Identification of option module
1020h	Identification of configuration
1200h	Server SDO parameters
1280h	Client SDO parameters
1400h - 14FFh	Reception PDO (RxPDO) communication parameters
1600h - 16FFh	Reception PDO (RxPDO) mapping parameters
1800h - 18FFh	Transmission PDO (TxPDO) communication parameters
1A00h - 1AFFh	Transmission PDO (TxPDO) communication parameters
1F22h	Concise DCF of network nodes
1F25h	Network node configuration request
1F80h	NMT Startup
1F81h	Network node assignment
1F82h	NMT command request
1F84h	Network node identification, device type
1F85h	Network node identification, manufacturer ID
1F86h	Network node identification, product code
1F87h	Network node identification, revision number
1F88h	Network node identification, serial number

<b>Index</b>	<b>Object contents/Meaning</b>
<b>1F89h</b>	Wait time for network node boot procedure
<b>2003h</b>	User device name
<b>2004h</b>	Synchronization with PLC
<b>2005h</b>	Reaction to PLC status
<b>2020h</b>	Network bootup setting
<b>2022h</b>	SDO wait time during the boot procedure of a node
<b>2023h</b>	Wait time for restoring the default settings
<b>2024h</b>	Wait time after network reset before network scanning begins
<b>2025h</b>	Wait time until the next node scan begins
<b>2100h</b>	Process map configuration
<b>2101h</b>	Asynchronous and synchronous areas in process map
<b>2102h</b>	Configured asynchronous objects in process map
<b>2103h</b>	Configured synchronous objects in process map

### B.3.2 Objects of the CANopen master pursuant to CiA DS 301 and DSP 302

#### 1000h - Device type

Specifies the supported device profile of the CANopen master and additional information.

Index	Subindex	Name	Data type	Access	Default value	Map-able
1000h	0h	device type	Unsigned32	ro	no	No

The 32-bit value is divided into two 16-bit values:

Byte:	MSB	LSB
Con- tents	Additional information, manufacturer-specific	Device profile
Value	0000h	0195h

The CANopen master supports the device profile CiA DSP 405 V 2.0 "Interface and Device Profile for IEC 61131-3 Programmable Devices". Additional information is not available.

#### 1001h - Error register

Contains a bit list with active errors of the CANopen master.

Index	Subindex	Name	Data type	Access	Default value	Map-able
1001h	0h	error register	Unsigned8	ro	no	No

The individual bits of the 8-bit value have the following meaning:

Bit	Meaning	Category
0	Generic error	Mandatory
1	Current	Optional
2	Voltage	Optional
3	Temperature	Optional
4	Communication error	Optional
5	Device profile specific	Optional
6	Reserved	Optional
7	Manufacturer specific	Optional

If a bit is set, the error is active. This object corresponds to the *error register* of the emergency telegram and thus contains the value of the *error register* from the last emergency



telegram sent by the CANopen master. Since the CANopen master does not generate emergency telegrams itself, the object 1001h only specifies the errors set by the user via the COM405\_SEND\_EMCY function module.

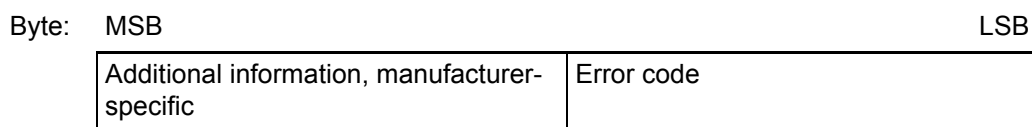
**1003h - Error memory**

Contains an error list of the CANopen master starting with the newest.

Index	Subindex	Name	Data type	Access	Default value	Map-able
1003h	-	pre-defined error field	Array	-	-	-
	0h	number of errors	Unsigned8	rw	0	No
	1h - 5h	standard error field	Unsigned32	ro	no	No

Subindex 0h contains the number of entered errors. The error list can be deleted by writing the value 0 to subindex 0h. Subindex 1h contains the current error. If a new error occurs, it is entered at subindex 1h and all others move down one subindex. A fault history of max. five entries is constructed.

The 32-bit value of the standard error field is divided into two 16-bit values:



The 16-bit error code is yielded after the *Emergency Error Code* is defined in the emergency telegram:

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

Error code	Meaning
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

The 16-bit additional information for the CANopen master contains the first two bytes of the *Manufacturer specific Error Field* from the emergency telegram.

Since the CANopen master does not generate emergency telegrams itself, the object 1003h only specifies the errors generated by the user via the COM405\_SEND\_EMCY function module.

#### 1005h - SYNC telegram COB-ID

Contains the COB-ID of the SYNC telegram and specifies whether the CANopen master generates the SYNC telegram.

Index	Subindex	Name	Data type	Access	Default value	Map-able
1005h	0h	COB-ID SYNC	Unsigned32	rw	40000080h	No

The individual bits of the 32-bit value have the following meaning:

Bit	Value/Meaning
31 (MSB)	Not relevant
30	0: CANopen master does not generate SYNC telegram 1: CANopen master generates SYNC telegram With CANopen master, generation of the SYNC telegram is activated by default.
29	0: 11-bit CAN ID 1: 29-bit CAN ID (not supported by optional CANopen master module)
28 - 11	Bits 28 - 11 with 29-bit CAN ID (not supported by optional CANopen master module)
10 - 0 (LSB)	COB-ID of the SYNC telegram. Since the optional CANopen master module uses the COB-ID after predefined issuing of the identifier, 80h is used by default.

Please be aware that further configuration of the SYNC mechanisms is carried out with object 2004h.

#### 1006h - SYNC interval

Length of the SYNC interval in  $\mu$ s.

Index	Subindex	Name	Data type	Access	Default value	Map-able
1006h	0h	Communication cycle period	Unsigned32	rw	0	No

The object specifies the time interval during which the CANopen master sends the SYNC telegram. The specified value in  $\mu$ s is converted to the internal resolution of 1 ms by the CANopen master, i.e. it is divided by 1,000. The result is evaluated as a whole number. SYNC intervals in multiples of 1 ms are therefore possible. If the object 1006h has the value 0, no SYNC telegram is generated and no data is exchanged with synchronous PDOs. Writing the object is not possible in the OPERATIONAL state.

#### NOTE



The synchronization type used with the PLC is specified via object 2004h. Object 2004h must be configured before object 1006h!

#### 1008h - Manufacturer device name

Contains the device name of the optional CANopen master module issued by the Baumüller company.

Index	Subindex	Name	Data type	Access	Default value	Map-able
1008h	0h	manufacturer device name	VISIBLE STRING	ro	no	No

The object specifies the name of the optional module as a character string, depending on the type of execution, e.g. "BMC-M-CAN-04".

### 1009h - Manufacturer hardware version

Contains the number of the hardware version of the optional CANopen master module issued by the Baumüller company.

Index	Subindex	Name	Data type	Access	Default value	Map-able
1009h	0h	manufacturer hardware version	VISIBLE STRING	ro	no	No

The object specifies the number of the hardware version of the optional module as a character string, depending on the type of execution, e.g. "3.0107".

### 100Ah - Manufacturer software version

Contains the number of the software version of the optional CANopen master module issued by the Baumüller company.

Index	Subindex	Name	Data type	Access	Default value	Map-able
100Ah	0h	manufacturer software version	VISIBLE STRING	ro	no	No

The object specifies the number of the software version of the optional module as a character string, depending on the type of execution, e.g. "6.1307.0201".

### 1010h - Save parameters

Enables the saving of object contents in nonvolatile memory. The object contents remain intact when the power is switched off and are reloaded when the power is switched on. The object contents are also loaded from the nonvolatile memory of the CANopen master after a RESET.

Index	Subindex	Name	Data type	Access	Default value	Map-able
1010h	-	store parameters	Array	-	-	-
	0h	largest subindex supported	Unsigned8	ro	no	No
	1h	save all parameters	Unsigned32	rw	no	No

Index	Subindex	Name	Data type	Access	Default value	Map-able
	2h	save communication parameters	Unsigned32	rw	no	No
	4h	save manufacturer defined	Unsigned32	rw	no	No

By writing the command "save" in ASCII code to the respective subindex, the contents of a specific selection of objects is saved. The "save" command is represented by the following 32-bit value:

Byte:   MSB LSB

e	v	a	s
65h	76h	61h	73h

The scope of object selection is yielded from the subindex:

Subindex	Memory scope
1	Saves the contents of all objects of the CANopen master
2	Saves the contents of the objects 1000h to 1FFFh of the CANopen master
4	Saves the contents of the objects 2000h to 5FFFh of the CANopen master

Reading a subindex 1 - 4 provides the following information:

Bit	Value/Meaning
31 - 2	<i>Reserved</i>
1	0: The device does not save the parameters automatically (e.g. when it is switched off) 1: The device saves the parameters automatically (e.g. when it is switched off)
0	0: Parameters cannot be saved by command 1: Parameters can be saved by command

### 1011h - Load default parameters

Enables activation of default parameters. The object contents saved in the nonvolatile memory are deleted.

Index	Subindex	Name	Data type	Access	Default value	Map-able
1011h	-	restore parameters	Array	-	-	-
	0h	largest subindex supported	Unsigned8	ro	no	No
	1h	restore all parameters	Unsigned32	rw	no	No

## B.3 CANopen objects of the CANopen masters (software version $\geq$ 01.20)

Index	Subindex	Name	Data type	Access	Default value	Map-able
	2h	restore communication parameters	Unsigned32	rw	no	No
	4h	restore manufacturer defined	Unsigned32	rw	no	No

By writing the command "load" in ASCII code to the respective subindex, a specific selection of object contents and their default values are loaded and activated after the next RESET of the CANopen master. The object contents in nonvolatile memory are deleted. The "load" command is represented by the following 32-bit value:

Byte:   MSB LSB

d	a	o	l
64h	61h	6Fh	6Ch

The scope of object selection is yielded from the subindex:

Subindex	Memory scope
1	Loads default values of all objects of the CANopen master
2	Loads default values of the objects 1000h to 1FFFh of the CANopen master
4	Loads default values of the objects 2000h to 5FFFh of the CANopen master

Reading a subindex 1 - 4 provides the following information:

Bit	Value/Meaning
31 - 1	Reserved
0	0: Default parameters cannot be activated 1: Default parameters can be activated

### 1014h - EMCY telegram COB-ID

Contains the COB-ID of the emergency telegram and specifies whether the EMCY telegram is valid.

Index	Subindex	Name	Data type	Access	Default value	Map-able
1014h	0h	COB-ID Emergency message	Unsigned32	rw	80h + node ID	No

The individual bits of the 32-bit value have the following meaning:

Bit	Value/Meaning
31	0: EMCY telegram is valid/activated 1: EMCY telegram is invalid/deactivated
30	<i>Reserved (always 0)</i>
29	0: 11-bit CAN ID 1: 29-bit CAN ID (not supported by optional CANopen master module)
28 - 11	Bits 28 - 11 with 29-bit CAN ID (not supported by optional CANopen master module)
10 - 0 (LSB)	COB-ID of the EMCY telegram. Since the optional CANopen master module uses the COB-ID after predefined issuing of the identifier, 80h + node ID is used by default. If a new COB-ID is issued, the node ID is no longer taken into account!

The EMCY telegram can only be generated by the user via the COM405\_SEND\_EMCY function module.

### 1016h - Heartbeat times of network nodes

Contains a list with the heartbeat times of the individual network nodes to be monitored by the CANopen master.

Index	Subindex	Name	Data type	Access	Default value	Map-able
1016h	-	Consumer Heartbeat Time	Array	-	-	-
	0h	number entries	Unsigned8 (1 – 127)	ro	no	No
	1h - 7Fh	Consumer Heartbeat Time	Unsigned32	rw	no	No

The object 1016h contains the expected heartbeat times of the individual network nodes in list form in the subindices 1 - 127. This list contains consecutive "Consumer Heartbeat Time" entries consisting of node number and the accompanying heartbeat time. A maximum of 127 nodes can be monitored. The "Consumer Heartbeat Time" entry has the following format:

Bits:	31 - 24	23 - 16	15 - 0
Contents	<i>Reserved (= 0)</i>	Node ID	Heartbeat time in ms

The heartbeat time in ms is converted to the internal resolution of 10 ms by the CANopen master, i.e. it is divided by 10. The result is evaluated as a whole number. Monitoring times in multiples of 10 ms are therefore possible. If a heartbeat time of 0 is entered, monitoring is not carried out.

CANopen master begins monitoring a node with the first heartbeat received. If a heartbeat is not received from a node within the specified time, the CANopen master can respond to this. The type of reaction is determined by objects 1F80h and 1F81h. It makes sense to set the heartbeat time on the network node to be monitored (object 1017h of the network node) lower than with the CANopen master in object 1016h.

### 1017h - Heartbeat time of CANopen master

Specifies the heartbeat time of the CANopen master.

Index	Subindex	Name	Data type	Access	Default value	Map-able
1017h	0h	Producer Hearbeat Time	Unsigned16	rw	0	No

The CANopen master sends heartbeat telegrams at the "Producer Hearbeat Time" time interval set in this object. The heartbeat time is specified in ms, whereby only multiples of 10 ms are possible, i.e. 10 ms, 20 ms, 30 ms etc.. Intermediate values are rounded up internally. If the time interval is 0, no heartbeat is sent. Heartbeat transmission begins as soon as a value different from zero is entered.

### 1018h - Identification of option module

Specifies the option module.

Index	Subindex	Name	Data type	Access	Default value	Map-able
1018h	-	Identity Object	Record	-	-	-
	0h	number of entries	Unsigned8	ro	no	No
	1h	Vendor ID	Unsigned32	ro	no	No
	2h	Product code	Unsigned32	ro	no	No
	3h	Revision number	Unsigned32	ro	no	No
	4h	Serial number	Unsigned32	ro	no	No

The object 1018h contains data for more precise identification of the optional CANopen master module. Subindices 1 - 3 have the following meaning:

Subindex	Name	Meaning
1	Vendor ID	Manufacturer ID. The Baumüller Nürnberg GmbH company has the ID 346
2	Product code	Not used, 0
3	Revision number	Not used, 0
4	Serial number	Serial number of the optional module



**1020h - Identification of configuration**

Enables the identification of the configuration of the CANopen master via the date and time.

Index	Subindex	Name	Data type	Access	Default value	Map-able
<b>1020h</b>	-	Verify Configuration	Array	-	-	-
	0h	number of entries	Unsigned8	ro	no	No
	1h	Configuration date	Unsigned32	rw	no	No
	2h	Configuration time	Unsigned32	rw	no	No

The object 1020h contains the date and time for more precise identification of the configuration of the CANopen master. A date can be entered in subindex 1 by the configuration tool or user, and a time can be entered in subindex 2. The contents of subindices 1 and 2 are deleted by CANopen master as soon as a new object is written. Exceptions include objects 1010h, 1020h, 1F25h, 1F82h and the objects of the process image (A000h etc.); writing of these objects does not lead to deletion of the contents of subindices 1 and 2.

Once the date and time are entered, the entire configuration is to be saved via object 1010h.

In greater detail, subindices 1 - 3 have the following meaning:

Subindex	Name	Meaning
1	Configuration date	The number of days that have passed since January 1st, 1984 is to be entered for the configuration date
2	Configuration time	The number of milliseconds that have passed since midnight is to be entered for the configuration date

**1200h - Server SDO parameters**

Contains the parameters for the access to the object directory of the CANopen master via SDO.

Index	Subindex	Name	Data type	Access	Default value	Map-able
<b>1200h</b>	-	Server SDO parameters	Record	-	-	-
	0h	number of entries	Unsigned8	ro	no	No
	1h	COB-ID Client -> Server (rx)	Unsigned32	ro	600h + node ID	No
	2h	COB-ID Server -> Client (tx)	Unsigned32	ro	580h + node ID	No

This object contains the parameters for SDO access to the object directory of the CANopen master. The CANopen master is the server in this case. The client can be another network node or the function modules for SDO communication from the xx.xx library if the master itself is accessed. The entries in subindices 1 and 2 have the following structure:

Bit	Value/Meaning
31	0: SDO is valid/activated 1: SDO is invalid/deactivated
30	<i>Reserved (always 0)</i>
29	0: 11-bit CAN ID 1: 29-bit CAN ID (not supported by optional CANOpen master module)
28 - 11	Bits 28 - 11 with 29-bit CAN ID (not supported by optional CANOpen master module)
10 - 0 (LSB)	COB-ID of the SDO telegram. Since the optional CANOpen master module uses the COB-ID after predefined issuing of the identifier, 600h + node ID is used for the client -> server direction and 580h + node ID for the server -> client direction. The node ID is the node number of the CANOpen master.

### 1280h - Client SDO parameters

The CANOpen master manages SDO access to the object directories of other network nodes internally. Configuration of the parameters of this object (e.g. COB-ID) by the user is not possible. The COB-IDs result from issuing via object 1200h, i.e. for inquiries via SDOs sent to a network node, 600h + node ID is used, and COB-ID 580h + node ID is the awaited answer.

### 1400h to 14FFh - Reception PDO (RxPDO) communication parameters

The objects 1400h to 14FFh determine the properties for transmission of RxPDO 1 to RxPDO 256 of the CANOpen master. Object 1400h describes RxPDO 1, object 1401h describes RxPDO 2, object 14FFh describes RxPDO 256 etc..

Index	Subindex	Name	Data type	Access	Default value	Map-able
<b>1400h</b>	-	RxPDO 1 parameters	Record	-	-	-
	0h	Largest subindex supported	Unsigned8	ro	2	No
	1h	COB-ID Receive RxPDO 1	Unsigned32	rw	NODEID+0x200	No
	2h	Transmission type RxPDO 1	Unsigned8	rw	255	No

Index	Subindex	Name	Data type	Access	Default value	Map-able
<b>1401h</b>	-	RxPDO 2 parameters	Record	-	-	-
	0h	Largest subindex supported	Unsigned8	ro	2	No
	1h	COB-ID Receive RxPDO 2	Unsigned32	rw	NODEID+0x300	No
	2h	Transmission type RxPDO 2	Unsigned8	rw	255	No

Index	Subindex	Name	Data type	Access	Default value	Map-able
<b>1402h</b>	-	RxPDO 3 parameters	Record	-	-	-
	0h	Largest subindex supported	Unsigned8	ro	2	No
	1h	COB-ID Receive RxPDO 3	Unsigned32	rw	NODEID+0x400	No
	2h	Transmission type RxPDO 3	Unsigned8	rw	255	No

Index	Subindex	Name	Data type	Access	Default value	Map-able
<b>1403h</b>	-	RxPDO 4 parameters	Record	-	-	-
	0h	Largest subindex supported	Unsigned8	ro	2	No
	1h	COB-ID Receive RxPDO 4	Unsigned32	rw	NODEID+0x500	No
	2h	Transmission type RxPDO 4	Unsigned8	rw	255	No

Index	Subindex	Name	Data type	Access	Default value	Map-able
<b>1404h – 14FFh</b>	-	RxPDO 5 – 256 parameters	Record	-	-	-
	0h	Largest subindex supported	Unsigned8	ro	2	No
	1h	COB-ID Receive RxPDO 5 – 256	Unsigned32	rw	0x80000000	No
	2h	Transmission type RxPDO 5 – 256	Unsigned8	rw	-	No

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

Subindex 1, bit	Value/Meaning
31 (MSB)	0: PDO exists and is valid 1: PDO does not exist or is invalid Only the first four RxPDOs of the CANopen master are activated. All other RxPDOs are deactivated. If more than four RxPDOs are used, the COB-IDs for the additional RxPDOs themselves must be issued (bits 10 - 0 in this subindex). For this reason, bit 31 of the RxPDOs 5 - 256 has a value of 1 and is to be set to 0 by the user as soon as the user issues a COB-ID.
30	0: The RxPDO can be requested via a remote request 1: The RxPDO cannot be requested via a remote request
29	0: 11-bit CAN ID 1: 29-bit CAN ID (not supported by optional CANopen master module)
28 - 11	Bits 28 - 11 with 29-bit CAN ID (not supported by optional CANopen master module)
10 - 0 (LSB)	COB-ID of the RxPDO. The first four RxPDOs have a default setting.

The transmission type of the RxPDO is specified with subindex 2:

Subindex 2, value	Type	Meaning
0	<i>Not supported</i>	
1 – 240	Synchronous	RxPDO with appropriate COB-ID received before the last SYNC telegram is accepted
254	Asynchronous	Each RxPDO with an appropriate COB-ID is accepted upon reception
255	Asynchronous	Each RxPDO with an appropriate COB-ID is accepted upon reception

### 1600h to 16FFh - Reception PDO (RxPDO) mapping parameters

The objects 1600h bis 16FFh determine the mapped objects of RxPDO 1 to RxPDO 256 of the CANopen master. Object 1600h describes RxPDO 1, object 1601h describes RxPDO 2, object 16FFh describes RxPDO 256 etc..

Index	Subindex	Name	Data type	Access	Default value	Map-able
<b>1600h - 16FFh</b>	-	RxPDO mapping	Record	-	-	-
	0h	Number of mapped objects	Unsigned8	rw	0	No
	1h - 8h	Object to be mapped	Unsigned32	rw	0	No

**Subindex 0** specifies the number of mapped objects in the RxPDO. As soon as the mapping of a RxPDO is changed, the number of mapped objects must first be set to zero. The mapping can then be set, and the number of mapped objects in subindex 0 must finally be entered. The mapping information of the objects in the RxPDO are specified in **subindices 1h to 8h**. Subindex 1h contains the information for the first mapped object, subindex 2h for the second mapped object etc.. A maximum of 8 (8h) objects can be mapped, whereby the maximum data range of 8 bytes of the PDO may not be exceeded. The contents of the 32-bit value of **subindices 1h to 8h** have the following meaning:

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 objects are mapped in an RxPDO one after another.

**1800h to 18FFh - Transmission PDO (TxPDO) communication parameters**

The objects 1800h to 18FFh determine the properties for transmission of TxPDO 1 to TxPDO 256. Object 1800h describes TxPDO 1, object 1801h describes TxPDO 2, object 18FFh describes TxPDO 256 etc..

Index	Subindex	Name	Data type	Access	Default value	Map-able
<b>1800h</b>	-	TxPDO 1 parameters	Record	-	-	-
	0h	Largest subindex supported	Unsigned8	ro	5	No
	1h	COB-ID Receive TxPDO 1	Unsigned32	rw	NODEID+0x180	No
	2h	Transmission type TxPDO 1	Unsigned8	rw	255	No
	5h	Event Timer TxPDO 1	Unsigned16	rw	0	No

Index	Subindex	Name	Data type	Access	Default value	Map-able
<b>1801h</b>	-	TxPDO 2 parameters	Record	-	-	-
	0h	Largest subindex supported	Unsigned8	ro	5	No
	1h	COB-ID Receive TxPDO 2	Unsigned32	rw	NODEID+0x280	No
	2h	Transmission type TxPDO 2	Unsigned8	rw	255	No
	5h	Event Timer TxPDO 2	Unsigned16	rw	0	No

Index	Subindex	Name	Data type	Access	Default value	Map-able
<b>1802h</b>	-	TxPDO 3 parameters	Record	-	-	-
	0h	Largest subindex supported	Unsigned8	ro	5	No
	1h	COB-ID Receive TxPDO 3	Unsigned32	rw	NODEID+0x380	No

## B.3 CANopen objects of the CANopen masters (software version $\geq$ 01.20)

Index	Subindex	Name	Data type	Access	Default value	Map-able
	2h	Transmission type TxPDO 3	Unsigned8	rw	255	No
	5h	Event Timer TxPDO 3	Unsigned16	rw	0	No

Index	Subindex	Name	Data type	Access	Default value	Map-able
<b>1803h</b>	-	TxPDO 4 parameters	Record	-	-	-
	0h	Largest subindex supported	Unsigned8	ro	5	No
	1h	COB-ID Receive TxPDO 4	Unsigned32	rw	NODEID+0x480	No
	2h	Transmission type TxPDO 4	Unsigned8	rw	255	No
	5h	Event Timer TxPDO 4	Unsigned16	rw	0	No

Index	Subindex	Name	Data type	Access	Default value	Map-able
<b>1804h - 18FFh</b>	-	TxPDO 5 - 256 parameters	Record	-	-	-
	0h	Largest subindex supported	Unsigned8	ro	5	No
	1h	COB-ID Receive TxPDO 5 – 256	Unsigned32	rw	0x80000000	No
	2h	Transmission type TxPDO 5 – 256	Unsigned8	rw	-	No
	5h	Event Timer TxPDO 5 – 256	Unsigned16	rw	0	No

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

Subindex 1, bit	Value/Meaning
31 (MSB)	0: PDO exists and is valid 1: PDO does not exist or is invalid Only the first four TxPDOs of the CANopen master are activated. All other TxPDOs are deactivated. If more than four TxPDOs are used, the COB-IDs for the additional TxPDOs themselves must be issued (bits 10 - 0 in this subindex). For this reason, bit 31 of the TxPDOs 5 - 256 has a value of 1 and is to be set to 0 by the user as soon as the user issues a COB-ID.
30	0: The PDO can be requested via a remote request 1: The PDO cannot be requested via a remote request
29	0: 11-bit CAN ID 1: 29-bit CAN ID (not supported by optional CANopen master module)

Subindex 1, bit	Value/Meaning
28 - 11	Bits 28 - 11 with 29-bit CAN ID (not supported by optional CANopen master module)
10 - 0 (LSB)	COB-ID of the TxPDO. The first four TxPDOs have a default setting.

The transmission type of the TxPDO is specified with **subindex 2**:

Subindex 2, value	Type	Meaning
0	<i>Not supported</i>	
1 – 240	Synchronous	Transmission occurs after transmission of the set number (1 - 240) of SYNC telegrams
254	Asynchronous	The TxPDO is sent as soon as the value of a mapped object has changed or a time event (expiration of an event time, subindex 5) has occurred.
255	Asynchronous	The TxPDO is sent as soon as the value of a mapped object has changed or a time event (expiration of an event time, subindex 5) has occurred.

The time in ms for the event-based PDO transfer of the time event is set in **subindex 5**. If the value is zero, no time event is triggered.

#### 1A00h to 1AFFh - Transmission PDO (TxPDO) mapping parameters

The objects 1A00h to 1AFFh determine the mapped objects of TxPDO 1 to TxPDO 256 of the CANopen master. Object 1A00h describes TxPDO 1, object 1A01h describes TxPDO 2, object 1AFFh describes TxPDO 256 etc..

Index	Subindex	Name	Data type	Access	Default value	Map-able
1A00h - 1AFFh	-	TxPDO mapping	Record	-	-	-
	0h	Number of mapped objects	Unsigned8	rw	0	No
	1h - 8h	Object to be mapped	Unsigned32	rw	0	No

**Subindex 0** specifies the number of mapped objects in the TxPDO. As soon as the mapping of a TxPDO is changed, the number of mapped objects must first be set to zero. The mapping can then be set, and the number of mapped objects in subindex 0 must finally be entered. The mapping information of the objects in the PDO are specified in **subindices 1h to 8h**. Subindex 1h contains the information for the first mapped object, subindex 2h for the second mapped object etc.. A maximum of 8 (8h) objects can be mapped, whereby the maximum data range of 8 bytes of the TxPDO may not be exceeded. The contents of the 32-bit value of **subindices 1h to 8h** have the following meaning:

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 objects are mapped in a TxPDO one after another.

### 1F22h - Concise DCF of network nodes

Contains a list with the "concise DCF" of the individual network nodes loaded on the individual nodes by the CANopen master during network bootup.

Index	Subindex	Name	Data type	Access	Default value	Map-able
1F22h	-	Concise DCF	Domain	-	-	-
	0h	number entries	Unsigned8 (1 – 127)	ro	127	No
	1h - 7Fh	Concise DCF nodes	Domain	rw	no	No

The object 1F22h contains the "concise DCF" of the individual network nodes in subindexes 1 - 127. A subindex is assigned to the number of the network node, i.e. the "concise DCF" of node 18 is in subindex 18, for example. A "concise DCF" (Device Configuration File) is a configuration loaded on the individual nodes by the CANopen master during network bootup. The configuration is described via a succession of objects with accompanying values. This configuration/the "concise DCF" has the following structure:

Entry	Data type
Number of objects	Unsigned32
Index 1	Unsigned16
Subindex 1	Unsigned8
Data length of the object in bytes	Unsigned32
Data	Domain
Index 2	Unsigned16
Subindex 2	Unsigned8
Data length of the object in bytes	Unsigned32
Data	Domain
Index 2	Unsigned16
Subindex 2	Unsigned8
Data length of the object in bytes	Unsigned32
Data	Domain



Entry	Data type
:	:
Index n	Unsigned16
Subindex n	Unsigned8
Data length of the object in bytes	Unsigned32
Data	Domain

The number of following objects is at the beginning of the configuration. This is followed by the individual objects with their index, subindex, data length and data. A maximum of 65,535 bytes can be entered in the subindices. This value applies for all subindices, and not for an individual one!

Example:

A sync interval of 10 ms (object 1006h), a guarding time of 250 ms (object 100Ch) and a life time factor of 4 (object 100Dh) are to be set on the node with number 18 during the network bootup. The data lengths of 32 bits, 16 bits and 8 bits result in accordance with CiA DS 301 (or the node manual). The following data in object 1F22h, subindex 18 of the CANopen master is then entered via an SDO transfer:

Byte	Value	Meaning
0	0x03	Number of objects
1	0x00	
2	0x00	
3	0x00	
4	0x06	Index object 1
5	0x10	
6	0x00	Subindex object 1
7	0x04	Data length object 1
8	0x00	
9	0x00	
10	0x00	
11	0x10	Data object 1
12	0x27	
13	0x00	
14	0x00	
15	0x0C	Index object 2
16	0x10	
17	0x00	Subindex object 2
18	0x02	Data length object 2
19	0x00	
20	0x00	
21	0x00	
22	0xFA	Data object 2
23	0x00	

Byte	Value	Meaning
24	0x0D	Index object 3
25	0x10	
26	0x00	Subindex object 3
27	0x01	Data length object 3
28	0x00	
29	0x00	
30	0x00	
31	0x04	Data object 3

The data length to be specified for SDO transfer is 32 bytes.

### 1F25h - Network node configuration request

Enables the requesting of configuration of network nodes by the CANOpen master during operation.

Index	Subindex	Name	Data type	Access	Default value	Map-able
1F25h	-	ConfigureSlave	ARRAY	-	-	-
	0h	number entries	Unsigned8	ro	128	No
	1h - 7Fh	Request re-configuration node	Unsigned32	rw	no	No
	80h	Request re-configuration all nodes	Unsigned32	rw	no	No

Configuration/reconfiguration of a network node by the CANOpen master during operation can be requested via object 1F25h without the need to perform a complete network boot-up. This may be necessary, for example, if new nodes are added to the network. Reconfiguration is triggered by the writing of the "conf" command to the respective subindex. Subindices 1 to 127 correspond to the number of node to be configured. If the "conf" command is written to subindex 128, all nodes are reconfigured. The "conf" command is represented by the following 32-bit value:

Byte:   MSB LSB

f	n	o	c
66h	6Eh	6Fh	63h

The node to be reconfigured may not be found in the OPERATIONAL state. If bit 2 for the respective network node (of corresponding subindex) in object 1F81h is set, the CANOpen master carries out a configuration automatically. Details are found in the description on object 1F81h.

**1F80h - NMT Startup**

Configures the startup behavior of the CANopen master.

Index	Subindex	Name	Data type	Access	Default value	Map-able
1F80h	0h	NMT Startup	Unsigned32	rw	0000000Dh	No

The individual bits of the 32-bit value have the following meaning:

Bit	Value/Meaning
0	0: <i>Not possible</i> 1: Optional CANopen master module is NMT master
1	0: After network bootup, switch only those nodes configured as slaves (object 1F81h, bit 0) to the "OPERATIONAL" state 1: After network bootup, switch all nodes to the "OPERATIONAL" state.
2	0: CANopen master switches to the "OPERATIONAL" state automatically after network bootup 1: CANopen master must be switched to the "OPERATIONAL" state by the application (e.g. via CIA405_NMT function module)
3	0: The slaves are switched to the "OPERATIONAL" state automatically after network bootup 1: The slaves must be switched to the "OPERATIONAL" state by the application (e.g. via CIA405_NMT function module)
4	0: When an error control event of a mandatory slave occurs (e.g. EMCY message, node guarding or heartbeat error) (see object 1F81h, bit 3), the slave is handled separately (see object 1F81h) 1: When an error control event of a mandatory slave occurs (e.g. EMCY message, node guarding or heartbeat error) (see object 1F81h, bit 3), all slaves are reset
5 - 31 (MSB)	<i>Reserved</i>

**1F81h - Network node assignment**

Contains a list of the network nodes assigned to the CANopen master and defines the behavior of the CANopen master in case of an error or similar event.

Index	Subindex	Name	Data type	Access	Default value	Map-able
1F81h	-	SlaveAssignment	Array	-	-	-
	0h	number entries	Unsigned8 (127)	ro	no	No
	1h - 7Fh	SlaveAssignment	Unsigned32	rw	no	No

The object 1F81h contains the assignment of individual network nodes for the CANopen master in list form in the subindices 1 - 127. The index in the list corresponds to the number of network nodes on the CANopen network.

In greater detail, the contents of a subindex are as follows:

Bit	Value/Meaning
0	0: The node with this number is not a node on the CANopen network, i.e. with nonexistent nodes, bit 0 must be 0 in the respective subindex. 1: The node with this number is a node on the CANopen network.
1	0: The application is informed of an "error control event" or other recognition of a booting slave 1: The application is informed of an "error control event" or other recognition of a booting slave, and the "error control service" is started automatically
2	0: In case of an "error control event" or other recognition of a booting slave, the slave is <b>not</b> automatically configured and started 1: In case of an "error control event" or other recognition of a booting slave, the slave is configured and started.
3	0: Optional slave: network is started even if this node does not report in 1: Mandatory slave: network is not started if this node does not report in
4,5,6	<i>Not supported</i>
7	<i>Reserved</i>
8 - 15 (byte 1)	Retry factor for node guarding. If a value is entered here, it is used for node guarding of the network node by the CANopen master
16 - 31 (bytes 2 + 3)	Guard time for node guarding. If a value is entered here, it is used for node guarding of the network node by the CANopen master. The guard time is specified in ms, whereby only multiples of 10 ms are possible, i.e. 10 ms, 20 ms, 30 ms etc.. Intermediate values are rounded up internally.

### 1F82h - NMT command request

Enables the requesting of the execution of NMT commands by the CANopen master during operation.

Index	Subindex	Name	Data type	Access	Default value	Map-able
1F82h	-	Request NMT	ARRAY	-	-	-
	0h	number entries	Unsigned8	ro	128	No
	1h - 7Fh	Request NMT	Unsigned8	rw	no	No
	80h	Request NMT all nodes	Unsigned8	wo	no	No

CANopen master can request the execution of an NMT command via object 1F82h to change a node state. The command is triggered by the writing of the requested command to the respective subindex. Subindices 1 to 127 correspond to the number of the node for which the command is valid. If the command is written to subindex 128, the command is valid for all nodes. The following commands are possible:

Node state	Command/Value
Stopped	4
Operational	5
Reset Node	6
Reset Communication	7
PreOperational	127

#### 1F84h - Network node identification, device type

Enables the specification of the expected device type of a network node upon network bootup.

Index	Subindex	Name	Data type	Access	Default value	Map-able
1F84h	-	Device Type Identification	ARRAY	-	-	-
	0h	number entries	Unsigned8	ro	127	No
	1h - 7Fh	Device Type Identification	Unsigned32	rw	no	No

When the network boots up, the CANopen master can check the device types, i.e. the contents of the object 1000h. The entries in object 1F84h are used for this purpose. Subindices 1 to 127 correspond to the number of the node for which the device type is checked. If a value is not entered in a subindex, checking does not occur. If a value is entered, it is compared to the contents of object 1000h on the network node. If they are different, an error message is sent (COM405\_NETWORK\_CONTROL function module).

#### 1F85h - Network node identification, manufacturer ID

Enables the specification of the expected vendor ID of a network node upon network bootup.

Index	Subindex	Name	Data type	Access	Default value	Map-able
1F85h	-	Vendor Identification	ARRAY	-	-	-
	0h	number entries	Unsigned8	ro	127	No
	1h - 7Fh	Vendor Identification	Unsigned32	rw	no	No

When the network boots up, the CANopen master can check the vendor IDs, i.e. the contents of the object 1018h (subindex 1). The entries in object 1F85h are used for this purpose. Subindices 1 to 127 correspond to the number of the node for which the vendor ID

is checked. If a value is not entered in a subindex, checking does not occur. If a value is entered, it is compared to the contents of object 1018h, subindex 1, on the network node. If they are different, an error message is sent (COM405\_NETWORK\_CONTROL function module). You must take into account that not all CANopen network nodes support object 1018h, which means that checking would not be possible.

### 1F86h - Network node identification, product code

Enables the specification of the expected product code of a network node upon network bootup.

Index	Subindex	Name	Data type	Access	Default value	Map-able
1F86h	-	Product Code	ARRAY	-	-	-
	0h	number entries	Unsigned8	ro	127	No
	1h - 7Fh	Product Code	Unsigned32	rw	no	No

When the network boots up, the CANopen master can check the product code, i.e. the contents of the object 1018h (subindex 2). The entries in object 1F86h are used for this purpose. Subindices 1 to 127 correspond to the number of the node for which the product code is checked. If a value is not entered in a subindex, checking does not occur. If a value is entered, it is compared to the contents of object 1018h, subindex 2, on the network node. If they are different, an error message is sent (COM405\_NETWORK\_CONTROL function module). You must take into account that not all CANopen network nodes support object 1018h, which means that checking would not be possible.

### 1F87h - Network node identification, revision number

Enables the specification of the expected revision number of a network node upon network bootup.

Index	Subindex	Name	Data type	Access	Default value	Map-able
1F87h	-	Revision Number	ARRAY	-	-	-
	0h	number entries	Unsigned8	ro	127	No
	1h - 7Fh	Revision Number	Unsigned32	rw	no	No

When the network boots up, the CANopen master can check the revision number, i.e. the contents of the object 1018h (subindex 3). The entries in object 1F87h are used for this purpose. Subindices 1 to 127 correspond to the number of the node for which the revision number is checked. If a value is not entered in a subindex, checking does not occur. If a value is entered, it is compared to the contents of object 1018h, subindex 3, on the network node. If they are different, an error message is sent (COM405\_NETWORK\_CONTROL function module). You must take into account that not all CANopen network nodes support object 1018h, which means that checking would not be possible.

**1F88h - Network node identification, serial number**

Enables the specification of the expected serial number of a network node upon network bootup.

Index	Subindex	Name	Data type	Access	Default value	Map-able
<b>1F88h</b>	-	Serial Number	ARRAY	-	-	-
	0h	number entries	Unsigned8	ro	127	No
	1h - 7Fh	Serial Number	Unsigned32	rw	no	No

When the network boots up, the CANopen master can check the serial number, i.e. the contents of the object 1018h (subindex 4). The entries in object 1F88h are used for this purpose. Subindices 1 to 127 correspond to the number of the node for which the serial number is checked. If a value is not entered in a subindex, checking does not occur. If a value is entered, it is compared to the contents of object 1018h, subindex 2, on the network node. If they are different, an error message is sent (COM405\_NETWORK\_CONTROL function module). You must take into account that not all CANopen network nodes support object 1018h, which means that checking would not be possible.

**1F89h - Wait time for network node boot procedure**

Length of the wait time for the boot procedure of "mandatory" network nodes in ms.

Index	Subindex	Name	Data type	Access	Default value	Map-able
<b>1F89h</b>	0h	Boot Time	Unsigned32	rw	0	No

The object specifies the time in ms that a bootup message of "mandatory" slaves (see object 1F81h) is awaited. If the CANopen master does not receive all bootup messages from "mandatory" slaves within this time after the network bootup is started, an error occurs (COM405\_NETWORK\_CONTROL function module). A time of 0 ms indicates that the CANopen master waits an infinite amount of time for the bootup message from a "mandatory" slave. The wait state is indicated at the COM405\_NETWORK\_CONTROL function module.

### B.3.3 Manufacturer-specific objects of the CANopen master

#### 2001h - Baud rate and node ID of the CANopen master

This object is not yet supported!

Determines the baud rate and node ID of the CANopen master. The values are used based on the settings of object 2002h. If object 2002h has a value of 0, the baud rate and node ID are set via an IEC 61131-3 function module and the values set in object 2001h are not taken into account.

Index	Subindex	Name	Data type	Access	Default value	Map-able
2001h	0h	Baud rate and Node ID	Unsigned32	rw	00 7F h	No

The contents of the 32-bit value of subindex 0h have the following meaning:

Byte:	MSB	LSB	
Contents	Reserved (16-bit)	Baud rate (8-bit)	Node ID (8-bit)
Range of values	-	0 - 8	1 - 127

The following baud rate values are permissible:

Baud rate value	Meaning
0	1 mbps
1	800 kbps
2	500 kbps
3	250 kbps
4	125 kbps
5	100 kbps
6	50 kbps
7	20 kbps
8	10 kbps

Default value for the baud rate is 0 (1 mbps). Default value for node ID is 127.

#### 2002h - Startup behaviour of CANopen master

This object is not yet supported!

Specifies the point to which the CANopen master starts up and then passes control on to IEC 61131-3 function modules.



Index	Subindex	Name	Data type	Access	Default value	Map-able
2002h	0h	Startup Control Master	Unsigned8	rw	0	No

The individual values have the following meaning:

Value	Meaning
0	Initialization and starting of the network bootup procedure are initiated by IEC 61131-3 function modules.
1	Initialization is carried out automatically. The contents of object 2001h are used for the baud rate and node ID of the CANopen master. Starting of the network bootup procedure is initiated by IEC 61131-3 function modules.
2	Initialization is carried out automatically. The contents of object 2001h are used for the baud rate and node ID of the CANopen master. The bootup procedure of the network is started off automatically.
3 - 255	<i>Reserved</i>

The set values are not valid until the device is switched off and on again. Manufacturer-specific objects must be saved via object 1010h whenever changes are made.

### 2003h - User device name

The user can issue his/her own device name with this object.

Index	Subindex	Name	Data type	Access	Default value	Map-able
2003h	0h	User Device Name	Visible String	rw	0	No

Any device name with max. 128 characters can be entered in **subindex 0h**.

### 2004h - Synchronization with PLC

Specifies the type of synchronization used with the PLC.

Index	Subindex	Name	Data type	Access	Default value	Map-able
2004h	0h	Synchronization PLC	Unsigned8	rw	0	No

The individual values have the following meaning:

Value	Meaning
0	Synchronization with the PLC does not occur.
1	The optional module generates an internal, synchronous IRP based on the "SNYC signal 1" signal, event 11 for event tasks under PROPROG wt II. The SYNC telegram is sent and the synchronous TxPDOs transferred with the generated, synchronous IRP. Ethernet is not deactivated.
2	The optional module generates an internal, synchronous IRP based on the "SNYC signal 2" signal, event 12 for event tasks under PROPROG wt II. The SYNC telegram is sent and the synchronous TxPDOs transferred with the generated, synchronous IRP. Ethernet is not deactivated.
3 - 255	<i>Reserved</i>
129	As with 1, except Ethernet is deactivated
130	As with 2, except Ethernet is deactivated
3 - 255	<i>Reserved</i>

If Ethernet is not deactivated (1, 2), precise synchronization is not possible!

### 2005h - Reaction to PLC status

Specifies how the CANopen master reacts to changes to the PLC state.

Index	Subindex	Name	Data type	Access	Default value	Map-able
2005h	0h	Reaction on PLC status	Unsigned8	rw	0	No

The individual values have the following meaning:

Value	Meaning
0	There is no reaction when the PLC exits the "Operation" (RUN) operating mode.
1	The optional CANopen master module switches to the "FATAL ERROR" state as soon as the PLC is not in the "Operation" (RUN) operating mode. The network is switched to the "STOPPED" state.
2 - 255	<i>Reserved</i>

The set values are not valid until the device is switched off and on again. Manufacturer-specific objects must be saved via object 1010h whenever changes are made.

### 2020h - Network bootup setting

The boot mechanism of the network can be influenced with this object.

Index	Subindex	Name	Data type	Access	Default value	Map-able
2020h	0h	Network Bootup Adjust	Unsigned32	rw	00000000h	No

The individual bits of the 32-bit value have the following meaning:

Bit	Value/Meaning
0	0: Presence of all possible nodes from 1 to 127 is scanned upon bootup of the network. 1: Only presence of nodes configured in object 1F81h is scanned upon bootup of the network. This setting is not recommended, as non-configured nodes can lead to malfunctions during operation.
1 - 31 (MSB)	<i>Reserved</i>

#### 2022h - SDO wait time during the boot procedure of a node

Length of the wait time in ms that the CANopen master waits for an SDO answer during the bootup procedure of a node.

Index	Subindex	Name	Data type	Access	Default value	Map-able
2022h	0h	Node Boot Wait SDO	Unsigned16	rw	50	No

Repeated SDO access to a node is necessary during its bootup procedure. All possible node numbers 1 - 127 are scanned, i.e. object 1000h is read. If a timeout occurs during the reply to this access, the node is viewed as not being present. This timeout is also determined by object 2022h.

If only b maXX 4400 slaves are found on the network, a time of 10 ms can be set.

#### 2023h - Wait time for restoring the default settings

Length of the wait time in ms that the CANopen master waits for a bootup message after a node reset before the DCF is downloaded.

Index	Subindex	Name	Data type	Access	Default value	Map-able
2023h	0h	Node Wait Reset	Unsigned16	rw	5000	No

Before the CANopen master loads the DCF to a node, it must load its default values when command to do so. The node must then be reset. Since the nodes require some time until they have performed a reset, the CANopen master must wait until it can begin downloading the DCF. This wait time is set via this object.

If only b maXX 4400 slaves are found on the network, a time of 50 ms can be set.

### 2024h - Wait time after network reset before network scanning begins

Length of the wait time in ms that the CANopen master waits until it begins scanning the network after a network reset.

Index	Subindex	Name	Data type	Access	Default value	Map-able
2024h	0h	Network Wait Start Scan	Unsigned16	rw	5000	No

The network is reset (reset communication) before it is scanned. Since the nodes require some time until they have performed a reset, the CANopen master must wait until it can begin scanning. This wait time is set via this object.

If only b maXX 4400 slaves are found on the network, a time of 50 ms can be set.

### 2025h - Wait time until the next node scan begins

If the CANopen master is missing a node, it cyclically checks whether the module is found on the network.

Index	Subindex	Name	Data type	Access	Default value	Map-able
2025h	0h	Node Wait Next Scan	Unsigned16	rw	2000	No

### 2100h - Process map configuration

Informs you of the configuration of the process map

Index	Subindex	Name	Data type	Access	Default value	Map-able
2100h	-	PI General Configuration	Array	-	-	-
	0	No. of entries supported	Unsigned8	ro	4	No
	1	Mode	Unsigned8	ro	1	No
	2	Version	Unsigned16	ro	100	No
	3	Size Outputs	Unsigned32	ro	2048	No
	4	Size Inputs	Unsigned32	ro	2048	No

**Subindex 1** specifies the mode of the process map. A value of 1 means that the segmented process is used to map the network variables. The segmented process requires configuration of the objects to be mapped (see object 2101h). **Subindex 2** reflects the version number of the process used. **Subindex 3** contains the sizes, in bytes, of the process map for outputs, i.e. data sent to the network by the application (network objects A000h - A279h). **Subindex 4** contains the sizes, in bytes, of the process map for inputs, i.e. data sent from the network to the application (network objects A480h - A6FFh).

### 2101h - Asynchronous and synchronous areas in process map

Determines the asynchronous and synchronous areas of the process map

Index	Subindex	Name	Data type	Access	Default value	Map-able
2101h	-	PI Asynchronous and Synchronous Area	Array	-	-	-
	0	No. of entries supported	Unsigned8	ro	4	No
	1	Start address and size asynchronous PI outputs	Unsigned32	rw	0000 0100h	No
	2	Start address and size asynchronous PI inputs	Unsigned32	rw	0000 0100h	No
	3	Start address and size synchronous PI outputs	Unsigned32	rw	0400 0100h	No
	4	Start address and size synchronous PI inputs	Unsigned32	rw	0400 0100h	No

The asynchronous and synchronous areas of the process map are determined in subindices 1 - 4 for both outputs and inputs. These areas contain objects transferred to asynchronous or synchronous PDOs. The objects of the process map themselves are determined via objects 2102h (asynchronous objects) and 2103h (synchronous objects).

The 32-bit value of each subindex is divided into two 16-bit values:

Byte:	MSB	LSB
Con- tents	Start address (byte address) of the area (16-bit)	Size of the area in bytes (16-bit)

The start address specified is relative to the start of the outputs/inputs of the process map. Byte addresses are specified. The size of an area is also specified in bytes. The required size of an area is yielded from the actual created objects in the process map (objects 2102h and 2103h), but multiples of 256 bytes are always to be issued.

You must comply with the following:

- The asynchronous area must be located ahead of the synchronous area
- The asynchronous area must begin at address 0. If there is no asynchronous area, the synchronous area can begin at address 0.
- The areas may not overlap.
- The permissible size of an area is a multiple of 256 bytes
- The size of the process map may not be exceeded on either side (outputs and inputs) (see object 2101h, subindices 3 and 4, for this purpose).

### 2102h - Configured asynchronous objects in process map

Enables the specification of asynchronous objects mapped in the process map.

Index	Subindex	Name	Data type	Access	Default value	Map-able
2102h	-	PI asynchronous Object Configuration	Array	-	Optional	-
	0	No. of entries supported	Unsigned8	ro	64	No
	1	1. configured object	Unsigned48	rw	A000 0000 0008h	No
	2	2. configured object	Unsigned48	rw	A040 0008 0018h	No
	3	3. configured object	Unsigned48	rw	A0C0 0020 0010h	No
	4	4. configured object	Unsigned48	rw	A100 0040 0010h	No
	5	5. configured object	Unsigned48	rw	A1C0 0060 0010h	No
	6	6. configured object	Unsigned48	rw	A200 00A0 0010h	No
	7	7. configured object	Unsigned48	rw	A480 0000 0008h	No
	8	8. configured object	Unsigned48	rw	A4C0 0008 0018h	No
	9	9. configured object	Unsigned48	rw	A540 0020 0010h	No
	10	10. configured object	Unsigned48	rw	A580 0040 0010h	No
	11	11. configured object	Unsigned48	rw	A640 0060 0010h	No
	12	12. configured object	Unsigned48	rw	A680 00A0 0010h	No
	13	13. configured object	Unsigned48	rw	0	No
	:	:	:	:	:	:
	64	64. configured object	Unsigned48	rw	0	No

**Subindex 0** contains the maximum possible number of configurable objects. The following **subindices 1 - 64** contain the configuration of the objects mapped in the process map. The objects are mapped in the asynchronous area of the process map, i.e. these objects are used for asynchronous PDOs.

The 48-bit value is divided into three 16-bit values:

Byte:   MSB LSB

Object index	Start address in DPRAM	Number of subindices used (max. 254)
--------------	------------------------	--------------------------------------

The object index specifies the index of the configured object. The following areas are permissible for the object index, based on the data type:

- Asynchronous objects written by the application:

Area	CANopen data type
A000h - A01Fh	Integer8
A040h - A05Fh	Unsigned8
A0C0h - A0DFh	Integer16
A100h - A11Fh	Unsigned16

Area	CANopen data type
A1C0h - A1DFh	Integer32
A200h - A21Fh	Unsigned32

- Asynchronous objects read by the application:

Area	CANopen data type
A480h - A49Fh	Integer8
A4C0h - A4DFh	Unsigned8
A540h - A55Fh	Integer16
A580h - A59Fh	Unsigned16
A640h - A65Fh	Integer32
A680h - A69Fh	Unsigned32

The start address in DPRAM is a byte address specified relative to the start of the process map outputs (A000h etc.) or inputs (A480h). The number of subindices used specifies the greatest subindex up to which objects of the specified object index are used.

#### NOTE



You must ensure that the address area of the asynchronous objects does not overlap the address area of the synchronous objects (object 2103h). To prevent this, the asynchronous objects should be created ahead of the synchronous objects.

### 2103h - Configured synchronous objects in process map

Enables the specification of synchronous objects mapped in the process map.

Index	Subindex	Name	Data type	Access	Default value	Map-able
2103h	-	PI synchronous Object Configuration	Array	-	Optional	-
	0	No. of entries supported	Unsigned8	ro	64	No
	1	1st configured object	Unsigned48	rw	A020 0400 0008h	No
	2	2nd configured object	Unsigned48	rw	A060 0408 0018h	No
	3	3rd configured object	Unsigned48	rw	A0E0 0420 0010h	No
	4	4th configured object	Unsigned48	rw	A120 0440 0010h	No
	5	5th configured object	Unsigned48	rw	A1E0 0460 0010h	No
	6	6th configured object	Unsigned48	rw	A220 04A0 0010h	No
	7	7th configured object	Unsigned48	rw	A4A0 0400 0008h	No
	8	8th configured object	Unsigned48	rw	A4E0 0408 0018h	No

Index	Subindex	Name	Data type	Access	Default value	Map-able
	9	9th configured object	Unsigned48	rw	A560 0420 0010h	No
	10	10th configured object	Unsigned48	rw	A5A0 0440 0010h	No
	11	11th configured object	Unsigned48	rw	A660 0460 0010h	No
	12	12th configured object	Unsigned48	rw	A6A0 04A0 0010h	No
	13	13th configured object	Unsigned48	rw	0	No
	:	:	:	:	:	:
	64	64th configured object	Unsigned48	rw	0	No

**Subindex 0** contains the maximum possible number of configurable objects. The following **subindices 1 - 64** contain the configuration of the objects mapped in the process map. The objects are mapped in the synchronous area of the process map, i.e. these objects are used for synchronous PDOs.

The 48-bit value is divided into three 16-bit values:

Byte: MSB

LSB

Object index	Start address in DPRAM	Number of subindices used (max. 254)

The object index specifies the index of the configured object. The following areas are permissible for the object index, based on the data type:

- Synchronous objects written by the application:

Area	CANopen data type
A020h - A03Fh	Integer8
A060h - A07Fh	Unsigned8
A0E0h - A0FFh	Integer16
A120h - A13Fh	Unsigned16
A1E0h - A1FFh	Integer32
A220h - A23Fh	Unsigned32

- Synchronous objects read by the application:

Area	CANopen data type
A4A0h - A4BFh	Integer8
A4E0h - A4FFh	Unsigned8
A560h - A57Fh	Integer16
A5A0h - A5BFh	Unsigned16
A660h - A67Fh	Integer32
A6A0h - A6BFh	Unsigned32



---

The start address in DPRAM is a byte address specified relative to the start of the asynchronous process map outputs (A000h etc.) or inputs (A480h). The number of subindices used specifies the greatest subindex up to which objects of the specified object index are used.



---

**NOTE**

You must ensure that the address area of the asynchronous objects (object 2101h) does not overlap the address area of the synchronous objects. The synchronous objects should always be created after the asynchronous objects. When issuing the start address of the first synchronous object, sufficient space for any additional necessary asynchronous objects should be available ahead of the address.

---





## Revision index

Revision level	State	Modifications
5.04031.03	27.10.2005	Application Manual extended with ProMaster and ProProg wt III new: Chapter 4.3 ProMaster, ProCANopen, ProProg wt III and Motion Control new: Chapter 4.4 ProMaster, ProCANopen and ProProg wt III Chapter 4.3 (old) now chapter 4.5 (new)
5.04031.04	29.01.2007	Revision of chapter 4.3 due to software update



## Revision index

---



# Index

## Numerics

100BaseTX	15, 16
10BaseT	15, 16

## A

Address classes	20, 22
ARP	28
ARP table	28

## B

Base address	40, 45
Basic address	104
Baud rate	104
Baumüller	7
Block SDO transfer	110
BMC-M-CAN-04	6
BMC-M-ETH-01	6
BMC-M-ETH-02	6
BMC-PLC-01	6
BMC-PSB-01	6
Bridge	19
Bus load	121, 134

## C

CAL	52
CAN	
Basics	51
CANop405_COB_ID	122
CANop405_EMERGENCY	150
CANop405_INIT	104
CANop405_NMT	107
CANop405_NODE_GUARDING	146
CANop405_PDO_INIT	122, 133
CANop405_PDO_READ	132
CANop405_PDO_WRITE	120
CANop405_SDO1_READ	112
CANop405_SDO1_WRITE	110
CANop405_SYNC	143
CANopen	
Basics	52
CANopen telegram structure	154
CANopen, general information	49
Checking the TCP/IP configuration	45
Class A	22
Class B	22
Class C	22
Class D	22
Class E	22
Client-server model	53
COB-ID	55
Commissioning	
Network	59, 102
Communication parameter	116
Communication profile	52

Configuration object	115
Create	
project	102
CSMA/CA method	51

## D

Data exchange with PDO	120
Data Length Code	154
Default mapping	120
Device profile	52
DLC (Data Length Code)	154

## E

Emergency	
Manufacturer-specific error code	149
Emergency Error Code	149
Emergency telegram structure	163
Emergency telegrams	148
Error Register	149
Ethernet	
General	15
Ethernet address	18
Ethernet data packet	18
ETHERNET_PLC_CONFIG_BMSTRUCT	40
ETHERNET_PLC_DIAG_BMSTRUCT	45
Event-controlled transmission	118
Example project	101
expedited SDO transfer	110

## F

Failure monitoring	145
First Steps	5
Fixed IP address	42

## G

Gateway	21
Default settings	21
Guard Time	145
Guarding functionality	146

## H

Hamming Distance	51
Host section	22
Hub	16

## I

Identifier	53
Identifier assignment, predefined	54
Inhibit time	118
INITIALIZATION	106
Internet link	21
InterNIC	22
Introduction	5



## Index

IP (Internet Protocol)	20	qualified	13
IP address	20	Ping	28
fixed	42	Plug-in connector	
Structure	20	Ethernet	16
Variable	43	Port numbers	28
IP useful data area	27	PRE-OPERATIONAL	106
ipconfig	32, 33, 35	Priority	56
<b>L</b>		Process Data Object	114
LED	105	Producer-consumer model	53
Life Guarding	145	Proxy	29
Life Time	145	<b>Q</b>	
<b>M</b>		Qualified Personnel	13
MAC address	18	<b>R</b>	
MAC-ID	18	Reading out node state	147
Mapping	119	Repeater	19
Default	120	Responsibility and liability	13
Media Access Control Identity	18	Rotary switches	40, 41
Modul number	103	Router	20, 21
Module		<b>S</b>	
change	28	Safety Instructions	7
Module ID	55	SDO (Service Data Object)	108
Module number	40, 45	SDO command specifier	155
Module numbers	104	SDO communication	110
Multiplexor	155	SDO telegram structure	155
<b>N</b>		SDO transfer	
Network adapter	30	expedited	110
Network Management	106	segmented	110
Network section	22	Segment	16
NMT (Network Management)	106	segmented SDO transfer	110
NMT command specifier	154	Service Data Object	108
NMT telegram structure	154	Session	27
Node Guarding	145, 146	Specialist	13
Node Guarding telegram structure	162	Star hub	16
<b>O</b>		State transition	106
Object	52	STOPPED	106
Object directory	108	Subnet mask	21
Object overview		Default settings	21
CANopen master	182	Switch	17
DS 301	167	SYNC telegram	140
OmegaOS data packet	29	SYNC telegram structure	161
OPERATIONAL	106	Synchronization window	142
<b>P</b>		Synchronizing data exchange	140
PDO		<b>T</b>	
Data exchange	120	TCP (Transport Control Protocol)	27
Event-controlled transmission	118	TCP/IP	19
Time-controlled transmission	118	Terms	
PDO (Process Data Object)	114	Definition	6
PDO telegram	114	Time Stamp	55
PDO telegram structure	160	Time-controlled transmission	118
Personnel	13	Toggle bit	145
		Transmission delay time	118



---

Transmission properties 115, 134, 140, 142  
Transmission Type 116  
Twisted-pair cable 16  
Two-wire cable 15

**V**

Variable IP address 43

**W**

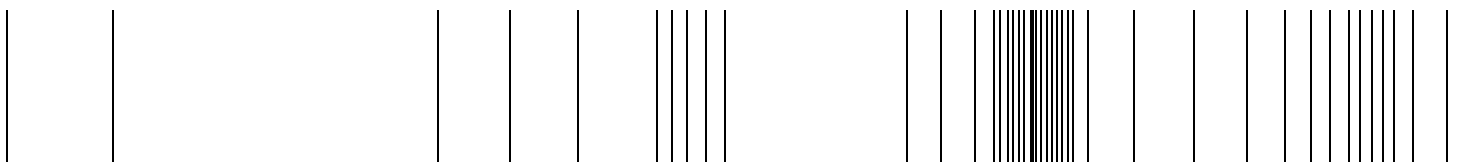
Warranty and Liability 14  
Windows 2000 33  
Windows 9x 36  
Windows NT 4.0 34  
Windows operating system 21, 30  
Windows XP 30  
winipcfg 37







**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](http://www.baumueller.de)

All the information in these Operating Instructions 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. Note that all the data/numbers/information that are quoted are current values at the time of printing. This information is not legally binding for dimensioning, calculation and costing. Before using the information listed in these Operating Instructions as the basis for your own calculations and/or applications, make sure that you have the latest most current information. This means that we accept no responsibility for the accuracy of the information.