BECKHOFF New Automation Technology

Manual | EN

# CX2500-x510

## CANopen Fieldbus Module


2024-11-20 | Version: 1.0

# Table of content

# 1 Foreword

## 1.1 Notes on the documentation

This description is only intended for the use of trained specialists in control and automation engineering who are familiar with applicable national standards.
It is essential that the documentation and the following notes and explanations are followed when installing and commissioning the components.
It is the duty of the technical personnel to use the documentation published at the respective time of each installation and commissioning.

The responsible staff must ensure that the application or use of the products described satisfy all the requirements for safety, including all the relevant laws, regulations, guidelines and standards.

### Disclaimer

The documentation has been prepared with care. The products described are, however, constantly under development.
We reserve the right to revise and change the documentation at any time and without prior announcement.
No claims for the modification of products that have already been supplied may be made on the basis of the data, diagrams and descriptions in this documentation.

### Trademarks

Beckhoff®, TwinCAT®, EtherCAT®, EtherCAT G®, EtherCAT G10®, EtherCAT P®, Safety over EtherCAT®, TwinSAFE®, XFC®, XTS® and XPlanar® are registered trademarks of and licensed by Beckhoff Automation GmbH.
Other designations used in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owners.

### Patent Pending

The EtherCAT Technology is covered, including but not limited to the following patent applications and patents:
EP1590927, EP1789857, EP1456722, EP2137893, DE102015105702
with corresponding applications or registrations in various other countries.

EtherCAT®

EtherCAT® is a registered trademark and patented technology, licensed by Beckhoff Automation GmbH, Germany

### Copyright

## 1.2 Safety instructions

### Safety regulations

Please note the following safety instructions and explanations!
Product-specific safety instructions can be found on following pages or in the areas mounting, wiring, commissioning etc.

### Exclusion of liability

All the components are supplied in particular hardware and software configurations appropriate for the application. Modifications to hardware or software configurations other than those described in the documentation are not permitted, and nullify the liability of Beckhoff Automation GmbH & Co. KG.

### Personnel qualification

This description is only intended for trained specialists in control, automation and drive engineering who are familiar with the applicable national standards.

### Description of symbols

In this documentation the following symbols are used with an accompanying safety instruction or note. The safety instructions must be read carefully and followed without fail!

**⚠ DANGER**

**Serious risk of injury!**

Failure to follow the safety instructions associated with this symbol directly endangers the life and health of persons.

**⚠ WARNING**

**Risk of injury!**

Failure to follow the safety instructions associated with this symbol endangers the life and health of persons.

**⚠ CAUTION**

**Personal injuries!**

Failure to follow the safety instructions associated with this symbol can lead to injuries to persons.

***NOTICE***

**Damage to the environment or devices**

Failure to follow the instructions associated with this symbol can lead to damage to the environment or equipment.

**Tip or pointer**

This symbol indicates information that contributes to better understanding.

## 1.3 Documentation issue state

| Version | Modifications |
| --- | --- |
| 1.0 | First version |

# 2 Product overview

## 2.1 Intended use

The CX20x0 device series is a modular control system designed for DIN rail installation. The system is scalable, so that the required modules can be assembled and installed in the control cabinet or terminal box as required.

**Only switch the PC off after closing the software**

Before the Embedded PC is switched off, the software currently running on it should be stopped properly in order to avoid data loss on the hard disk. Please read the section on “Switching off”.

Switch off all system components and uncouple the Industrial PC from the system if the PC is not used for control purposes, e.g. during a function test. To disconnect first pull the first terminal behind the power supply unit (optional), then pull the connectors of the fieldbus connections.
System components that have been switched off must be secured against being switched on again.

The Embedded PC’s power supply unit must be supplied with 24 $V_{DC}$.

**NOTICE**

**Damage to the environment or devices**

Do not exchange any parts when under power! The exchange of controller parts when live can lead to short-circuits or overvoltages. These can damage the controller itself and connected peripherals (terminals, monitors, input devices, etc.).

When components are being fitted or removed, the supply voltage must be switched off.

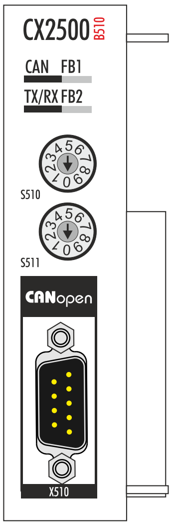
**Software knowledge**

**NOTICE**

**System malfunctions**

Mandatory software knowledge! Every user must be familiar with any of the functions of the software installed on the PC that he can reach.

## 2.2 Fieldbus module CX2500-M510/B510




**Fieldbus module**

The fieldbus modules are left-sided attachments for the CX2000 Embedded PC series. Up to four modules can be connected in any order.

### CANopen master (CX2500-M510)

The fieldbus module CX2500-M510 is a CANopen master and enables a segment-like construction of control structures in large plants and machines. Further Beckhoff fieldbus components such as Bus Couplers, Bus Terminal Controllers, drive components, etc. can be used with an Embedded PC for configuring control structures.

Fieldbus masters are used for decentralized collection of process data and signals in large machines and plants. The number of slaves that can be connected to the master is only limited by the respective bus system. Using master and slave connections makes it possible to link several Embedded PCs with each other via the fieldbus level.

The fieldbus modules are detected, parameterized and configured in TwinCAT, and the connected I/O components are added. TwinCAT is also used for diagnostics.

### CANopen slave (CX2500-B510)

The fieldbus module CX2500-B510 is a CANopen slave and enables an Embedded PC to be used as subordinate decentral controller for configuring complex or modular systems.

The CANopen slave receives external process data from the master and processes them or returns data from its own process periphery to the master after processing.

Like the CANopen master, the CANopen slave fieldbus module is parameterized and configured in TwinCAT.

### Functioning

CANopen is a widely used fieldbus system, which was developed by the CAN in Automation (CiA) association and in the meantime has been adopted for international standardization.

| Further Information |
| --- |
| CAN in Automation (CiA) website:<br>www.can-cia.org |

CANopen consists of the protocol definition (communication objects) and the device profiles.

The protocol definition includes the following communication objects:

- Network Management (NMT)

- Process Data Objects (PDO)
- Service Data Objects (SDO)
- And protocols with special functions

The device profiles standardize the data content for the respective device class. The term device class covers devices such as electric drives, I/O modules, sensors and controllers. The device profiles define the functionality and the structure of the object directory.

The device parameters and process data are stored in a structured object directory. Any data in this object directory is accessed via service data objects (SDO).

Process data objects (PDO) are used for fast communication of input and output data. There are, additionally, a few special objects (such as telegram types) for network management (NMT), synchronization, error messages and so on.

### Communication Types

CANopen defines several communication types for process data objects:

- Event driven:
  Telegrams are sent as soon as their contents have changed. This means that the process image as a whole is not continuously transmitted, only its changes.
- Cyclic synchronous:
  A SYNC telegram causes the modules to accept the output data that was previously received, and to send new input data.
- Requested (polled):
  A CAN data request telegram causes the modules to send their input data.

The required communication type is set via the transmission type parameter (see: PDO [▶ 43] tab).

### Bus access procedures

CAN operates based on the Carrier Sense Multiple Access (CSMA) method, i.e. all devices have equal rights and access the bus as soon as it is free (multi-master bus access). The exchange of messages is thus not device-oriented but message-oriented. This means that every message is unambiguously marked with a prioritized identifier.

In order to avoid collisions on the bus when messages are sent by different devices, a bit-wise bus arbitration is carried out at the start of the data transmission. The bus arbitration allocates bus bandwidth to the messages in order of priority. At the end of the arbitration phase only one bus devices occupies the bus; collisions are avoided, and the bandwidth is optimally utilized.

### Configuration and parameterization

The TwinCAT System Manager allows all the CANopen parameters to be set conveniently. An "eds" file (an electronic data sheet) is available on the Beckhoff website (www.beckhoff.de) for the parameterization of Beckhoff CANopen devices using configuration tools from other manufacturers.

### Certification

The Beckhoff CANopen devices have a powerful implementation of the protocol, and are certified by the CAN in Automation Association (www.can-cia.org).

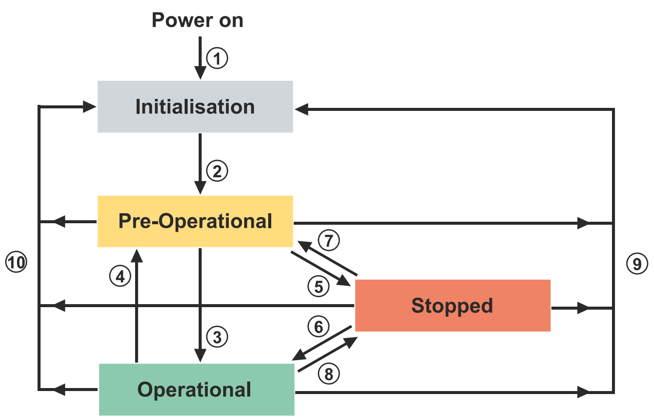
## 2.3 Network Management

The network management (NMT) defines the communication behavior of a CANopen device and consists of the states initialization, pre-operational, operational and stopped.

When a device is switched on or restarted, the device automatically switches to the initialization state. When the initialization state is completed, the device automatically switches to pre-operational state.

From this state any other state can be assumed. For sample, only a single CAN message is required to make the devices start:
Start_Remote_Node: Identifier 0, two data bytes: 0x01, 0x00.
This message transfers the devices to the operational state.

The following diagram shows which states a CANopen device can assume:




- **Initialization**
  The device automatically switches to the initialization state. When the initialization state is completed, the device automatically switches to pre-operational state.
- **Pre-Operational**
  After the initialization the device automatically switches to pre-operational state, i.e. without an external command. In this state the service data objects (SDO) are already active, and the device can be configured. The process data objects (PDO) are still locked.
- **Operational**
  In operational state the process data objects (PDO) are active.
  If the device is no longer able to set outputs, read inputs or communicate due to external influences (e.g. CAN fault, no output voltage) or internal influences (e.g. K-bus error), it tries to send a corresponding emergency message. The device then assumes error state and switches back to pre-operational state. This enables the NMT status machine of the master to detect fatal errors immediately.
- **Stopped**
  In stopped state (previously *prepared*) no communication with the device is possible. Only network management (NMT) messages are received. The outputs go into the fault state.

**State transitions**

State transitions are executed with a CAN message. The CAN messages have a very simple structure: CAN identifier 0, with two bytes of data content.

- The first data byte contains the command specifier (cs),
- the second data byte contains the node address (node ID); node address 0 addresses all nodes (broadcast).

| 11 bit identifier | 2 byte of user data | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0x00 | cs | Node ID | | | | | | |

The following table provides an overview of all possible state transitions and the corresponding command specifiers (cs). The diagram shown above, which illustrates the states, is part of the overview:

| State transition | Command Specifier cs | Explanation |
|---|---|---|
| (1) | -- | The initialization state is reached automatically at power-up |
| (2) | -- | After initialization the pre-operational state is reached automatically - this involves sending the boot-up message. |
| (3), (6) | cs = 1 = 0x01 | Start_Remote_Node.<br>Starts the device, enables outputs and starts the PDO transfer. |
| (4), (7) | cs = 128 = 0x80 | Enter_Pre-Operational.<br>Stops the PDO transmission, SDO still active. |
| (5), (8) | cs = 2 = 0x02 | Stop_Remote_Node.<br>Outputs go into the fault state, SDO and PDO switched off. |
| (9) | cs = 129 = 0x81 | Reset_Node.<br>Carries out a reset. All objects are reset to their power-on defaults. |
| (10) | cs = 130 = 0x82 | Reset_Communication.<br>Carries out a reset of the communication functions. Objects 0x1000 - 0x1FFF are reset to their power-on defaults. |

**Sample 1**
The following telegram puts all the modules in the network into the error state (outputs in a safe state):

| 11 bit identifier | 2 byte of user data | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0x00 | 0x02 | 0x00 | | | | | | |

**Sample 2**
The following telegram is used to reset (restart) node 17:

| 11 bit identifier | 2 byte of user data | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0x00 | 0x81 | 0x11 | | | | | | |

## 2.3.1 Boot-up message

After the initialization phase and the self-test the device sends the boot-up message, which is a CAN message with a data byte (0) on the identifier of the guarding or heartbeat message:
CAN-ID = 0x700 + node ID.

In this way temporary failure of a module during operation (e.g. due to a voltage interruption), or a module that is switched on at a later stage, can be reliably detected, even without Node Guarding. The sender can be determined from the message identifier (see default identifier allocation).

It is also possible, with the aid of the boot-up message, to recognize the nodes present in the network at start-up with a simple CAN monitor, without having to make write access to the bus (such as a scan of the network by reading out parameter 0x1000).

Finally, the boot-up message communicates the end of the initialization phase. The device indicates that it can now be configured or started.

### Format of the Boot-up message

| 11 bit identifier | 2 byte of user data | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0x700 (=1792)+ node ID | 0x00 | | | | | | | |

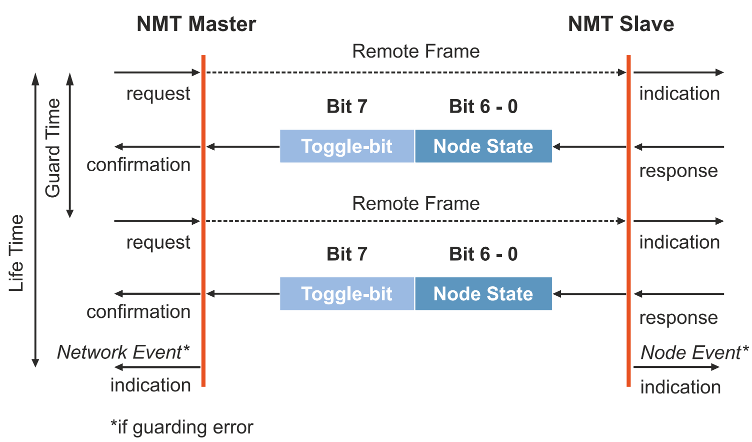
## 2.3.2 Node Monitoring

The heartbeat and guarding mechanisms are available for monitoring CANopen device failures. These mechanisms are particularly important for CANopen, since in the event-driven operating mode the devices do not report on a regular basis. In the case of "guarding", the devices are cyclically interrogated about their status by means of a data request telegram (remote frame), whereas with "heartbeat" the nodes transmit their status on their own initiative.

### Guarding: Node Guarding and Life Guarding

Node guarding is used to monitor the slaves, which themselves use life guarding to detect a faulty guarding master.

Node guarding involves the master issuing request telegrams, referred to as remote frames (remote transmit requests) to the guarding identifiers of the slaves to be monitored. The slaves respond with a guarding message. The message contains the node state of the slave and toggle bit, which has to change after each message. If the node state or the toggle bit do not match the values expected by the master or if there is a general lack of response, the master assumes an error.




At the time of the first guarding message of the slave the toggle bit has the value 0. Subsequently, the toggle bit alternates after each guarding message and enables lost messages to be detected. The remaining seven bits contain the node state, thereby transferring the slave status to the master.

| Node State | Status |
|---|---|
| 4 = 0x04 | Stopped |
| 5 = 0x05 | Operational |
| 127 = 0x7F | Pre-Operational |

**Sample**

The guarding message for node 27 (0x1B) must be requested by a remote frame having identifier 0x71B ($1819_{dec}$). If the node is Operational, the first data byte of the answer message alternates between 0x05 and 0x85, whereas in the Pre-Operational state it alternates between 0x7F and 0xFF.

**Guard time and life time factor**

If the master requests the guard messages in a strict cycle, the slave can detect the failure of the master. In this case, if the slave fails to receive a message request from the master within the set Node Life Time (a guarding error), the slave assumes that the master has failed (the watchdog function).

In this case the slave sets its outputs to error state, sends an emergency telegram and returns to pre-operational state. After a guarding time-out the procedure can be triggered again with a new guarding message.

The node life time is calculated based on the following parameters:

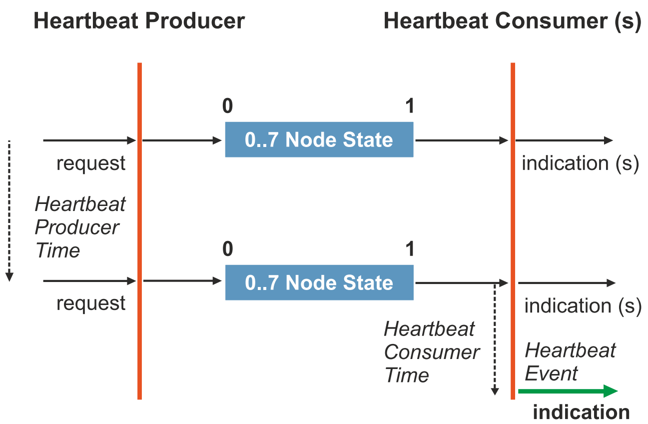
- guard time (object 0x100C) and
- life-time factor (object 0x100D)

Life time = guard time **x** life time factor

If one of the two parameters is 0 (default), the master is not monitored (no life guarding) (see: CAN node [▸ 40] tab).

**Heartbeat: Node Monitoring without Remote Frame**

If the heartbeat method is active, the devices automatically send their respective status messages cyclically. In contrast to the guarding method, the heartbeat method does not involve sending of remote frames, resulting in less bus load. The master also sends its heartbeat telegram cyclically, enabling the slaves to detect a master failure (see: CAN node [▸ 40] tab).




The toggle bit is not used with the heartbeat method, and the devices send their respective status cyclically.

## 2.3.3 Process Data Objects (PDO)

In many fieldbus systems the entire process image is continuously transferred - usually in a more or less cyclic manner. CANopen is not limited to this communication principle, since with CANopen the process data are transferred based on the producer/consumer model. It means that a device sends its process data spontaneously (producer), while all other devices listen and use a CAN identifier to decide whether the telegram is of interest for them, in which case they process it accordingly (consumer).

The process data in CANopen is divided into segments with a maximum of 8 bytes. These segments are known as process data objects (PDOs). The process data objects (PDOs) each correspond to a CAN telegram. They are allocated and prioritized based on a specific CAN identifier.

The process data objects (PDOs) are subdivided into receive PDOs (RxPDOs) and send PDOs (TxPDOs); the ID is based on the device perspective. A device sends its input data with TxPDOs and receives the output data in the RxPDOs.
This designation is retained in TwinCAT.

### Communication parameters

The process data objects (PDOs) can be assigned different communication parameters, as required. Like all CANopen parameters, these are also listed in the object directory of the device. The communication parameters can be accessed via service data objects (SDOs).

The parameters for the receive PDOs are contained in index 0x1400 (RxPDO1) to 0x15FF (RxPDO512), which means up to 512 RxPDOs can exist. In the same way, the entries for the transmit PDOs are located from index 0x1800 (TxPDO1) to index 0x19FF (TxPDO512).

For each existing process data object (PDO) there is an associated communication parameter object. TwinCAT automatically assigns the set parameters to the relevant object directory entries. These entries and their significance for the communication of process data are explained below.

### CAN identifier

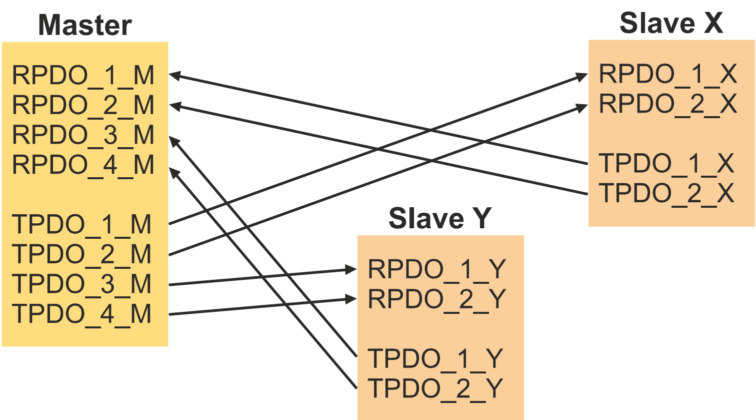
The main communication parameter for process data objects (PDOs) is the CAN identifier (also referred to a communication object identifier, COB ID). The CAN identifier is used to identify the CAN telegrams and determines their priority for bus access. For each CAN telegram there can only be one sender (producer). However, since CAN sends all messages based on the broadcast method, a telegram can be received by any number of devices (consumers), as described. In other words, a device can make its input information available to several devices at the same time, even without transfer by a logical master.

The CAN identifier is contained in subindex 1 of the communication parameter set. It is coded as a 32-bit value in which the least significant 11 bits (bits 0...10) contain the CAN identifier itself. The 32-bit object data width enables 29-bit CAN identifiers to be entered. The use of the 29-bit version is limited to special applications and is therefore not supported by the Beckhoff CANopen devices. The highest bit (bit 31) can be used to activate the process data object or to turn it off.

### PDO linking

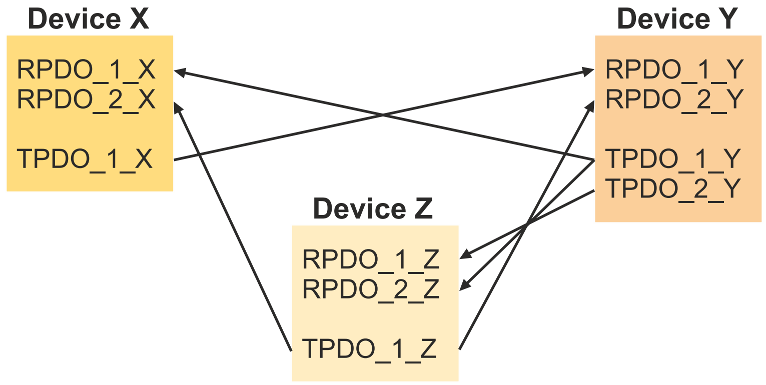
By default all devices (slaves) communicate with a central controller (master). By default no slave listens to the process data objects (PDOs) and therefore the CAN identifiers of another slave.

Standard communication between several slaves and a master:




To exchange process data objects (SDOs) directly between the devices without a master, the CAN identifiers have to be adjusted accordingly. The TxPDO identifiers of the producer must match the RxPDO identifier of the consumer. This procedure is known as PDO linking. It permits, for sample, easy construction of electronic drives in which several slave axes simultaneously listen to the actual value in the master axis TxPDO.

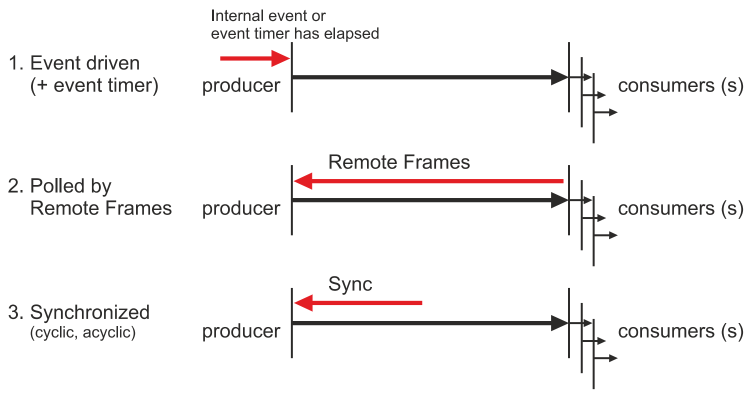
Communication without master based on PDO linking:




**PDO transmission types**

CANopen offers following options for transferring process data objects (PDOs):

- Event driven
- By polling
- Synchronized




- **Event driven:**
  If an input value changes, a device immediately transfers its process data objects (PDOs). This enables optimum utilization of the transmission bandwidth, since only the change in the process image is transferred, while the response time is short, since the devices do not wait for a query from a master if the input values change.

  As from CANopen Version 4 it is possible to combine the event driven type of communication with a cyclic update. The process data objects (in this case the TxPDOs) are then transferred once a set time (event timer) has elapsed. If an input value changes within the set time, the time (event timer) is reset.

  With RxPDOs the set time (event timer) is used to monitor the arrival of the event-driven process data objects (PDOs). A device switches to error state, if no process data objects (PDOs) arrived within the set time.

- **Polled:**
  The process data objects (PDOs) are polled through request telegrams (remote frames). In this way the process data objects (PDOs) are transferred even if there is no change, for sample if a monitor or diagnostic device is included in the network at runtime.

  Function blocks with integrated full message filtering (FullCAN) usually respond to a request telegram directly and immediately send the data contained in the corresponding send buffer. The application must ensure that the data are continuously updated. CAN controllers with simple message filtering (BasicCAN) on the other hand pass the request on to the application which can now compose the telegram with the latest data. This does take longer, but does mean that the data is up-to-date. Beckhoff use CAN controllers following the principle of Basic CAN.

Since this device behavior is usually not transparent to the user, and because there are CAN controllers still in use that do not support remote frames at all, polled communication will only with reservation be recommended for operative running.

- **Synchronized**
  It is not only for drive applications that it is worthwhile to synchronize the determination of the input information and the setting the outputs. For this purpose CANopen provides the SYNC object, a CAN telegram of high priority but containing no user data, whose reception is used by the synchronized users as a trigger for reading the inputs or for setting the outputs.

**PDO transmission types: Parameterization**

The PDO transmission type determines how the transfer of the process data objects (PDOs) is triggered and how the received process data objects (PDOs) are treated:

| Transmission type | Cyclical | Acyclical | Synchronous | Asynchronous |
|---|---|---|---|---|
| 0 | | X | X | |
| 1-240 | X | | X | |
| 254, 255 | | | | X |

The type of transmission is parameterized for RxPDOs in the objects at 0x1400ff, sub-index 2, and for TxPDOs in the objects at 0x1800ff, sub-index 2. In TwinCAT the PDO transmission type is set on the PDO tab under Transmission Type (see: PDO [▸ 43] tab).

- **Acyclic Synchronous:**
  PDOs of transmission type 0 function synchronously, but not cyclically. An RxPDO is only evaluated after the next SYNC telegram. In this way, for instance, axis groups can be given new target positions one after another, but these positions only become valid at the next SYNC - without the need to be constantly outputting reference points.
  In contrast, a TxPDO determines its input data after a SYNC telegram (synchronous process image) and forwards its input data if the input data have changed.
  Transmission type 0 therefore combines the send reason „synchronized“ with the send reason “event-driven”.

- **Cyclic Synchronous:**
  With transmission types 1-240 the PDO is sent cyclically after each n-th SYNC (n=1...240). In this way it is possible to parameterise a fast cycle for digital inputs (n=1), for sample, while the data of the analog inputs are transferred in a slower cycle (e.g. n=10). RxPDOs generally do not differentiate between the transmission types 0...240. A received PDO is set to valid with the next SYNC telegram. The cycle time (SYNC rate) can be monitored (object 0x1006), so that if the SYNC fails the device reacts in accordance with the definition in the device profile, and switches, for sample, its outputs into the fault state.

  The SYNC telegram is coupled with the link task, so that new input data are available with each task start. If a synchronous PDO fails to materialize, this is detected and reported to the application.

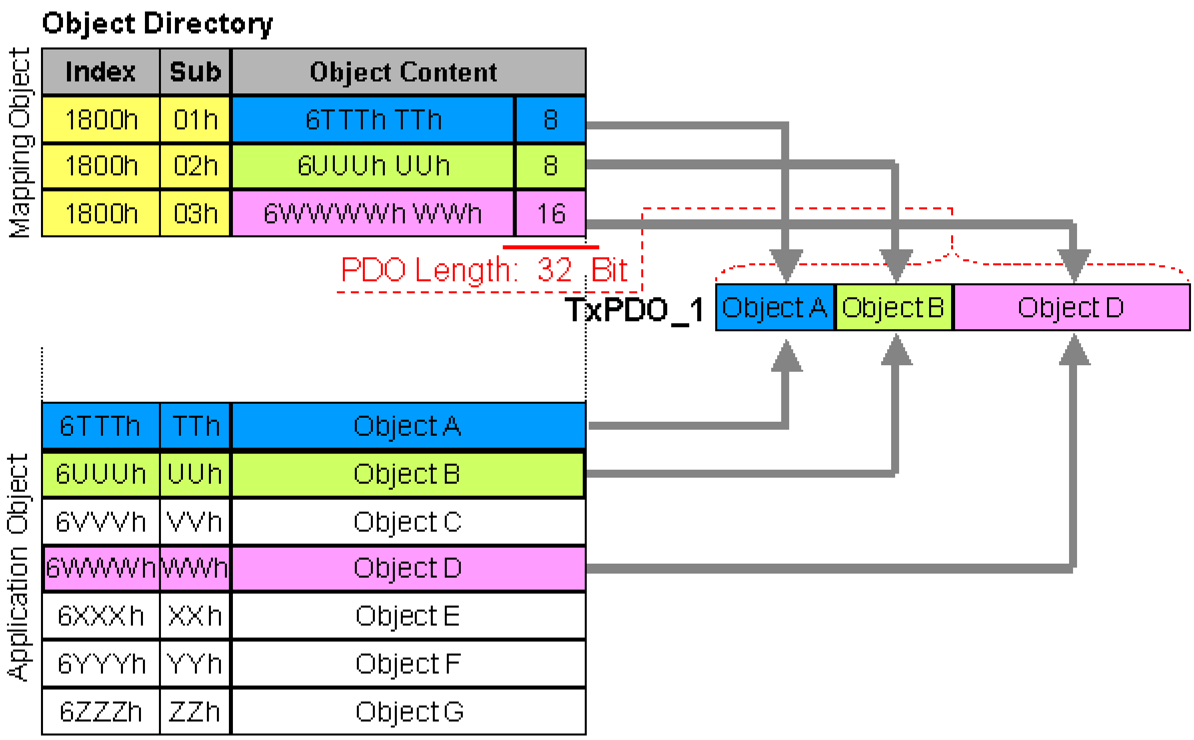
- **Asynchronous:**
  The transmission types 254 + 255 are asynchronous or event-driven. In transmission type 254, the event is specific to the manufacturer, whereas for type 255 it is defined in the device profile. In the simplest case, the event is the change of an input value - this means that every change in the value is transmitted. The asynchronous transmission type can be coupled with the event timer, thus also providing input data when no event has just occurred. Once the set time has elapsed, this is interpreted as an additional event, and the input data are sent.

### PDO Mapping

PDO mapping refers to mapping of the application objects (real time data) from the object directory to the process data objects. The CANopen device profile provide a default mapping for every device type, and this is appropriate for most applications. Thus the default mapping for digital I/O simply represents the inputs and outputs in their physical sequence in the transmit and receive process data objects.

The default PDOs for drives contain 2 bytes each of a control and status word and a set or actual value for the relevant axis.

The current mapping can be read by means of corresponding entries in the object directory. These are known as the mapping tables. The first location in the mapping table (sub-index 0) contains the number of mapped objects that are listed after it. The tables are located in the object directory at index 0x1600ff for the RxPDOs and at 0x1A00ff for the TxPDOs.




### Digital and analog input/output modules: Read out the I/O number

The current number of digital and analog inputs and outputs can be determined or verified by reading out the corresponding application objects in the object directory:

| Parameter | Object directory address |
|---|---|
| Number of digital input bytes | Index 0x6000, sub-index 0 |
| Number of digital output bytes | Index 0x6200, sub-index 0 |
| Number of analog inputs | Index 0x6401, sub-index 0 |
| Number of analog outputs | Index 0x6411, sub-index 0 |

### Variable mapping

As a rule, the default mapping of the process data objects already satisfies the requirements. For special types of application the mapping can nevertheless be altered: the Beckhoff CANopen Bus Couplers, for instance, thus support variable mapping, in which the application objects (input and output data) can be freely allocated to the PDOs. The mapping tables must be configured for this: as from Version 4 of CANopen, only the following procedure is permitted, and must be followed precisely:

1. First delete the PDO (set 0x1400ff, or 0x1800ff, sub-index 1, bit 31 to "1")
2. Set sub-index 0 in the mapping parameters (0x1600ff or 0x1A00ff) to "0"
3. Change mapping entries (0x1600ff or 0x1A00ff, SI 1..8)

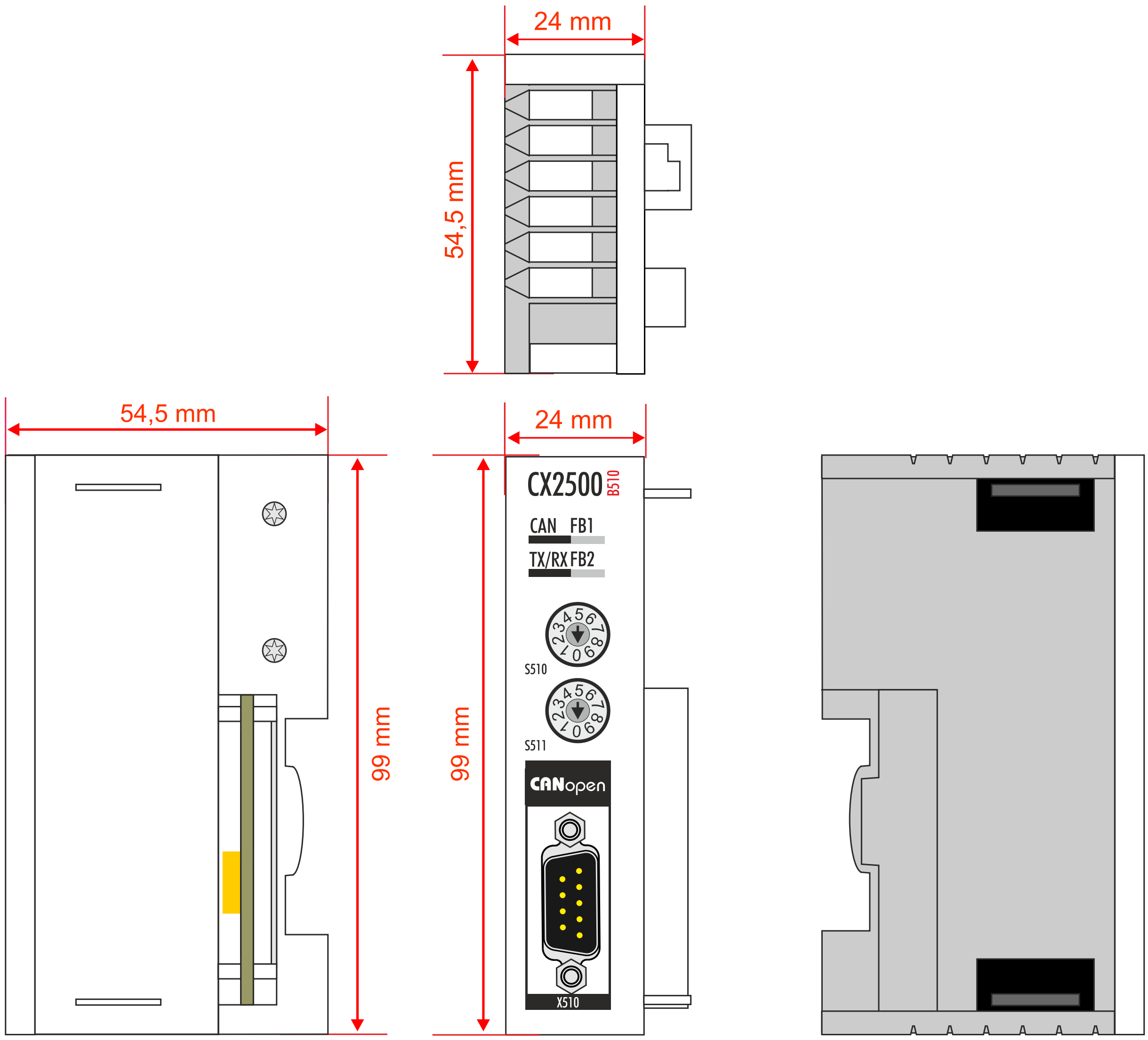
4. Set sub-index 0 in the mapping parameters to the valid value. The device then checks the entries for consistency.
5. Create PDO by entering the identifier (0x1400ff or 0x1800ff, sub-index 1).

**Dummy Mapping**

A further feature of CANopen is the mapping of placeholders, or dummy entries. The data type entries stored in the object directory, which do not themselves have data, are used as placeholders. If such entries are contained in the mapping table, the corresponding data from the device is not evaluated. In this way, for instance, a number of drives can be supplied with new set values using a single CAN telegram, or outputs on a number of nodes can be set simultaneously, even in event-driven mode.

## 2.4 Technical data - CANopen

**Dimensions**



### CX2500-M510

| Technical data | CX2500-M510 |
|---|---|
| Fieldbus | CANopen |
| Data transfer rates | 10, 20, 50, 100, 125, 250, 500, 800, 1.000 kBaud |
| Bus interface | 1 x D-sub socket, 9-pin |
| Bus devices | max. 64 |
| max. process image | 512 Tx PDOs / 512 Rx PDOs |
| Interface to the CPU | PCI-Express |
| Max. power loss | 2,8 W |
| Properties | CANopen – supported PD communication types: event-driven, time-controlled, synchronous, polling; emergency message handling, guarding and heartbeat, boot-up after DS302; online bus load monitor and bus trace, error management for each device freely configurable |
| Dimensions (W x H x D) | 24 mm x 99 mm x 54,5 mm |
| Weight | approx. 180 g |
| Operating/storage temperature | -25…+60 °C/-40…+85 °C |
| Relative humidity | 95 %, no condensation |
| Vibration/shock resistance | conforms to EN 60068-2-6/EN 60068-2-27 |
| EMC immunity/emission | conforms to EN 61000-6-2/EN 61000-6-4 |
| Protection class | IP 20 |

### CX2500-B510

| Technical data | CX2500-B510 |
|---|---|
| Fieldbus | CANopen |
| Data transfer rates | 10, 20, 50, 100, 125, 250, 500, 800, 1.000 kBaud |
| Bus interface | 1 x D-sub socket, 9-pin |
| Extendable process image | Up to 15 virtual slaves in addition |
| max. process image | 16 slaves x (16 Tx PDOs / 16 Rx PDOs (8 byte per PDO)) |
| Interface to the CPU | PCI-Express |
| Max. power loss | 2,8 W |
| Dimensions (W x H x D) | 24 mm x 99 mm x 54,5 mm |
| Weight | approx. 180 g |
| Operating/storage temperature | -25…+60 °C/-40…+85 °C |
| Relative humidity | 95 %, no condensation |
| Vibration/shock resistance | conforms to EN 60068-2-6/EN 60068-2-27 |
| EMC immunity/emission | conforms to EN 61000-6-2/EN 61000-6-4 |
| Protection class | IP 20 |

### Device Profile

The BECKHOFF CANopen devices support all types of I/O communication, and correspond to the device profile for digital and analog input/output modules (DS401 Version 1). For reasons of backwards compatibility, the default mapping was not adapted to the DS401 V2 profile version.

# 3 Unpacking and transport

**Short circuit due to moisture**

Moisture can lead to short circuits. Moisture can form during transport in cold weather or in the event of large temperature fluctuations.

Avoid moisture formation (condensation) in the device, and leave the device to adjust to room temperature slowly. If condensation has occurred, wait at least 12 hours before switching on the device.

**Proceed as follows to unpack the unit:**

1. Remove packaging.
2. Do not discard the original packaging. Keep it for transporting the device in the future.
3. Check the delivery for completeness by comparing it with your order.
4. Please keep the associated paperwork. It contains important information for handling the unit.
5. Check the contents for visible shipping damage.
6. If you notice any shipping damage or inconsistencies between the contents and your order, you should notify Beckhoff Service.

**Shipping and relocation**

Despite the robust design of the unit, the components are sensitive to strong vibrations and impacts. Therefore, during transport please protect your device from:

- mechanical stress and
- use the original packaging.

Follow the specified storage conditions for the device and store it at temperatures between -40 °C and +85 °C.

# 4 Mounting and wiring

## 4.1 Installing the extension module at the CX20x0 system

**Damage to the contacts**

Incorrect installation may result in damage to the contacts of the fieldbus modules.

The fieldbus modules may only be attached on the left-hand side of the basic CPU module.

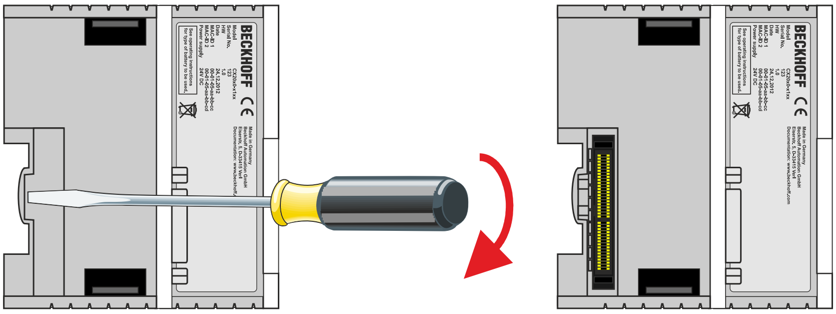
A maximum of four fieldbus modules can be attached to the basic CPU module via the internal PCI Express bus. In delivery state the fieldbus module is equipped with a protective cap for the bus connector. Before continuing, connect the fieldbus module to the basic CPU module.

Prerequisites for this step:

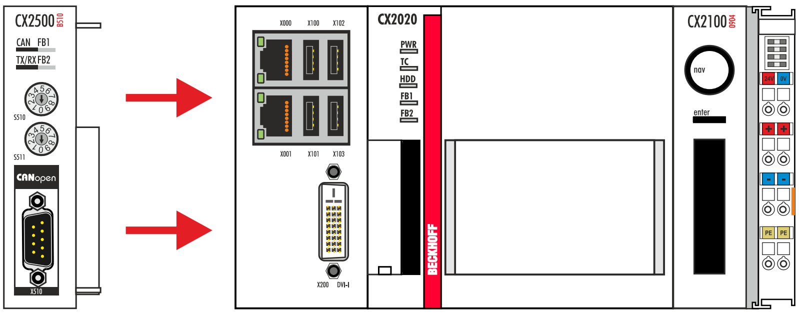
- Exit the software and shut down the CX20x0 hardware configuration.
- Switch off the power supply.

**Install the fieldbus module as follows:**

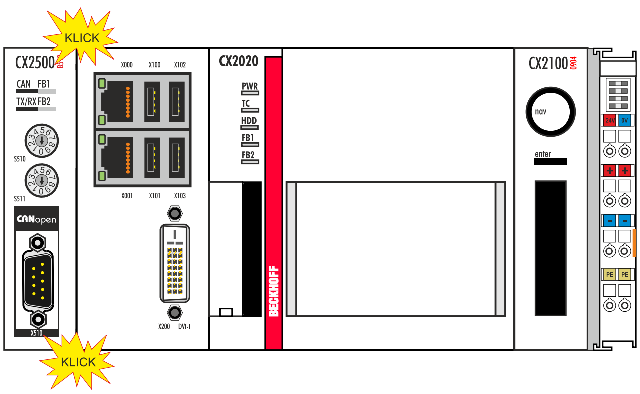
1. Put a screwdriver into the recess on the short side of the protective cap and lever it off.




2. Remove the protective cap to expose the bus connector.
3. Insert the fieldbus module on the **left-hand side of** the basic CPU module.




4. The module clicks into the basic CPU module.



⇨ The fieldbus module has been installed successfully, if the individual modules were connected straight and flush.
Next, you can install the bar clips, thereby reinforcing interlocking of the modules.

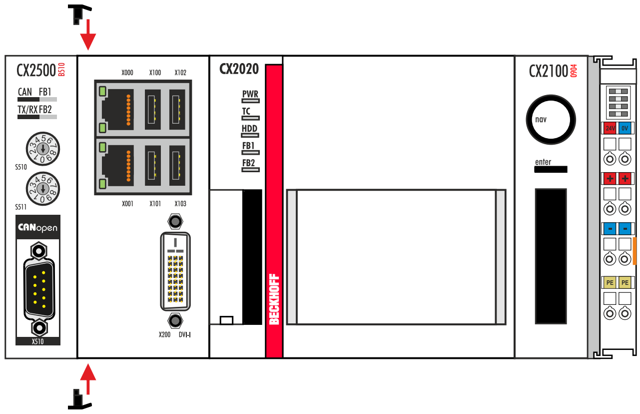
# 4.2 Installing the bar clips

**Installing the bar clips**

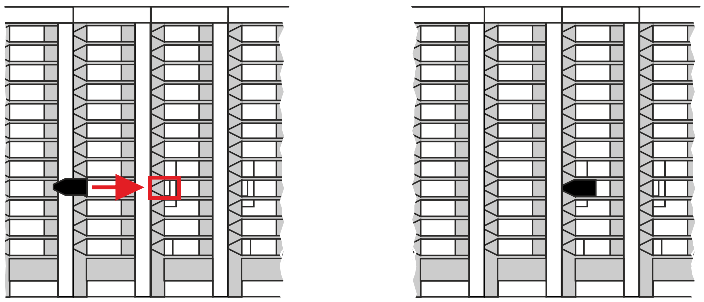
Usually, the connection between the modules is strong enough. However, the basic CPU module and the fieldbus modules may be subjected to shocks, vibrations or other impacts. The modules can be securely connected to one another with the aid of bar clips.

**Install the bar clips as follows:**

1. Attach the bar clips to the top and bottom of the devices.




2. Push the bar clips between the cooling fins of your devices, as shown in the image below.



⇨ The bar clips have been installed successfully, if they don't protrude and are level with the cooling fins of your modules.
Once all fieldbus modules are locked, the whole assembly can be installed on the mounting rail.

## 4.3 Installation on the mounting rail

**Installation position and minimum distances**

The modules or devices may overheat, if the installation position is incorrect or the minimum distances are not adhered to.

The devices may only be operated at ambient temperatures up to 60 °C. Ensure adequate ventilation. Choose a horizontal installation position for the devices. Leave at least 30 mm clearance above and below the installed devices.

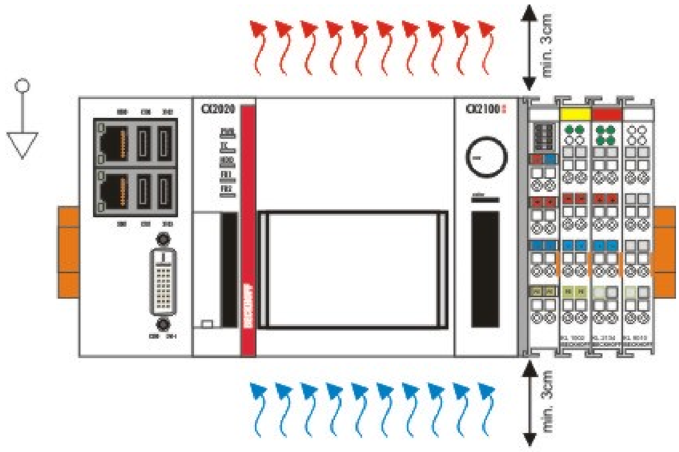
**Correct installation position**

The devices may be operated up to an ambient temperature of 60 °C. The high performance and the compact design of the CX2000 Embedded PC series may result in increased heat generation. The heat is dissipated via a passive ventilation system. Venting of the devices requires a correct installation position.

Ventilation openings are located at the top and bottom of the housing. The system therefore has to be installed horizontally. This ensures optimum airflow through the devices in vertical direction.

During installation, leave at least 30 mm clearance above and below the devices to ensure adequate ventilation.

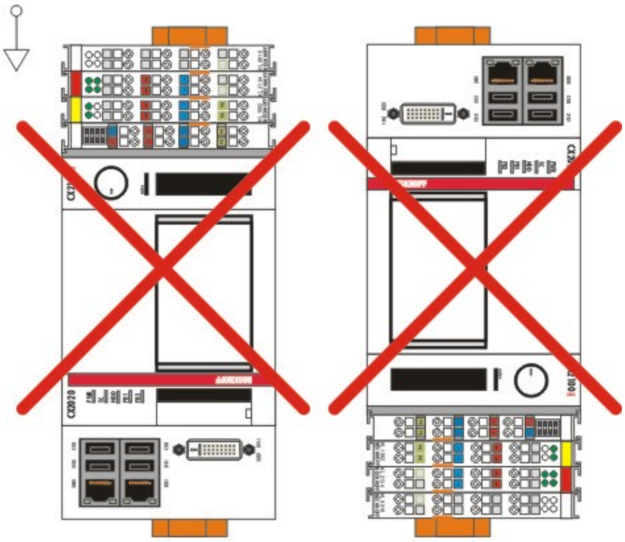
The images below show the permitted and two unacceptable installation positions:

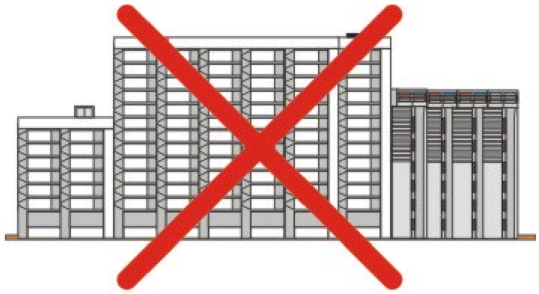



gültige Installationslage
valid position

**Incorrect installation positions**

The CX20x0 system must not be operated vertically on the DIN rail. This installation position provides insufficient ventilation for the devices. In horizontal position the devices are not sufficiently ventilated either.





ungültige Installationslage
invalid position

ungültige Installationslage
invalid position

**Installation on the mounting rail**

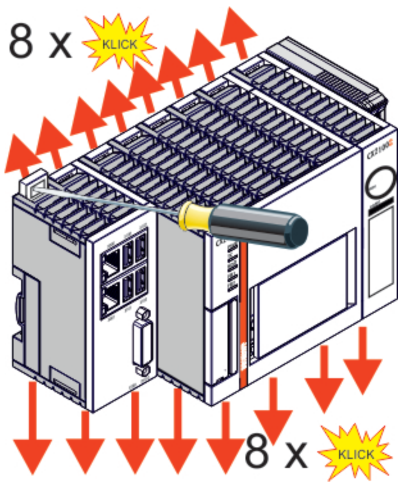
**Avoid damage**

Do not use force or excessive pressure when installing the devices on the mounting rail.

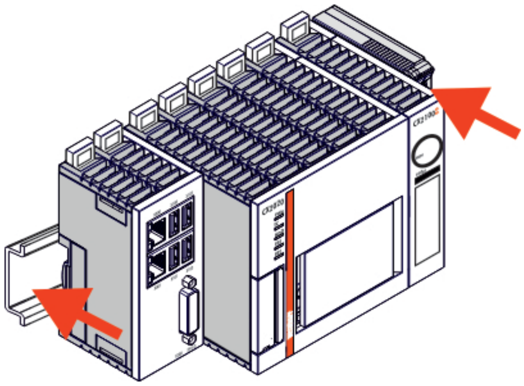
The CX20x0 can easily be installed on the mounting rail. The housing is designed such that it can be pushed against the mounting rail and engaged on it.

**Install the devices on the mounting rail as follows:**

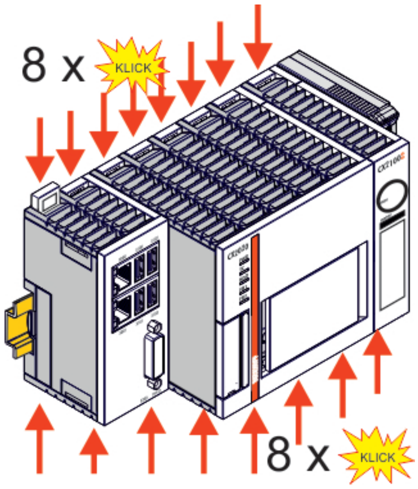
1. Unlock the latches at the top and bottom.



2. Position the devices at the front of the mounting rail. Gently push the devices onto the mounting rail until you can hear a click and the devices are engaged.

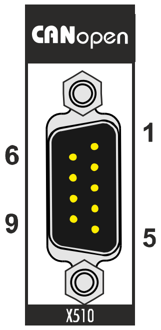


3. Then lock the latches again.




⇨ The devices are now installed successfully. Verify that the devices are installed correctly and that all devices are engaged on the mounting rail.
In the next step you can commission the devices.

## 4.4 CANopen connection

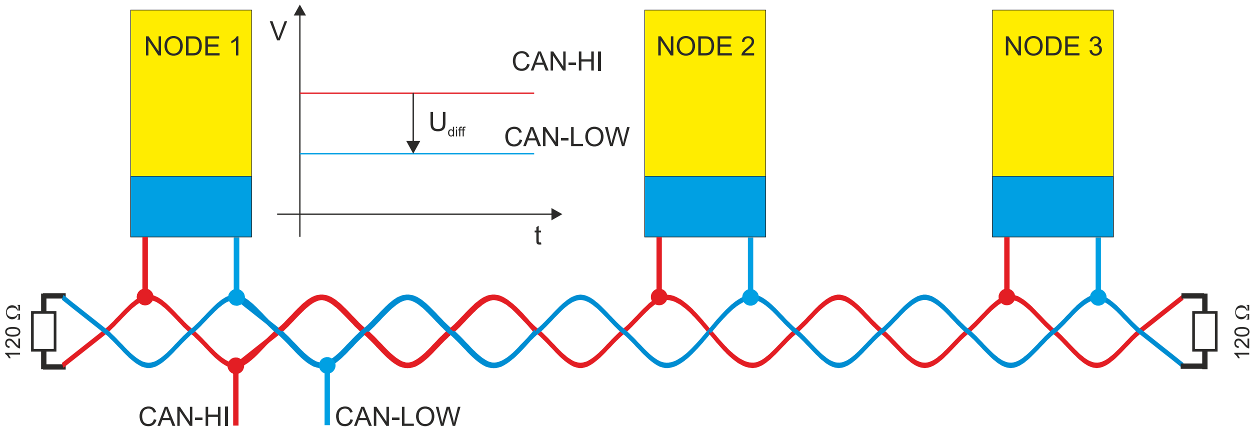



The CAN bus line is connected via a 9-pin D-sub socket with the following configuration:

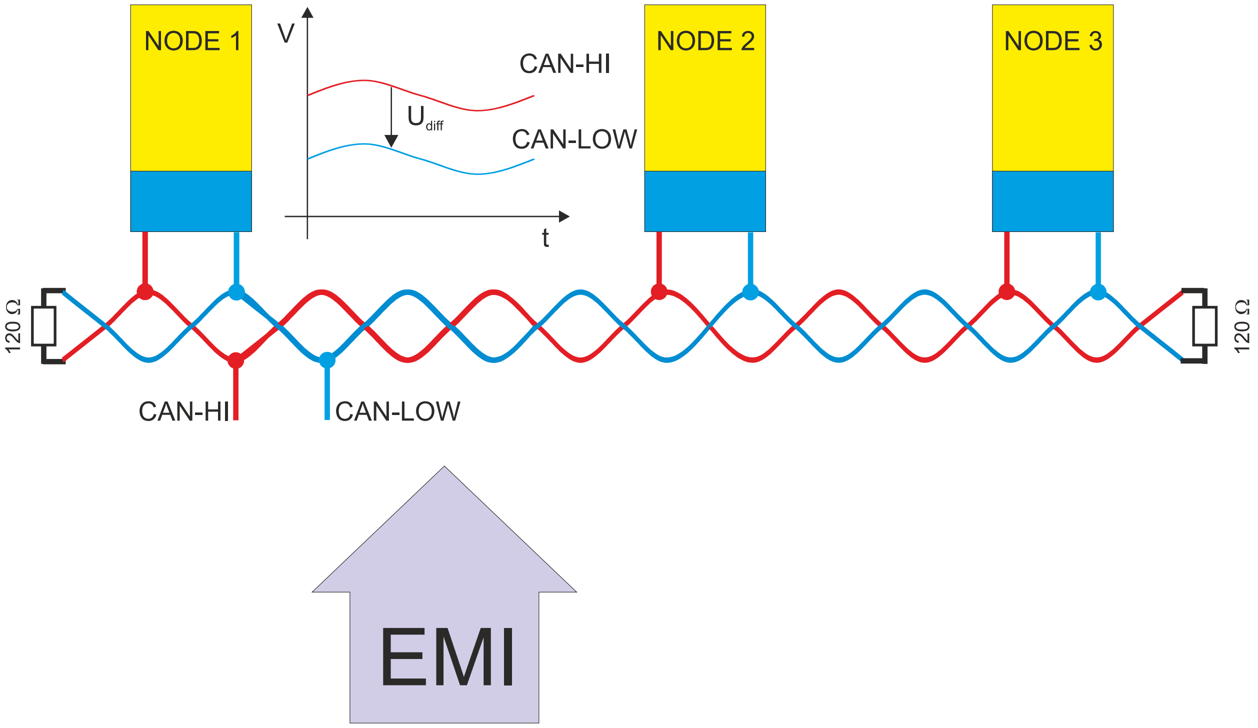
| Pin | Assignment |
| --- | --- |
| 1 | not used |
| 2 | CAN low (CAN-) |
| 3 | CAN ground (internally connected to pin 6) |
| 4 | not used |
| 5 | Shield |
| 6 | CAN ground (internally connected to pin 3) |
| 7 | CAN high (CAN+) |
| 8 | not used |
| 9 | not used |

## 4.5 Cabling

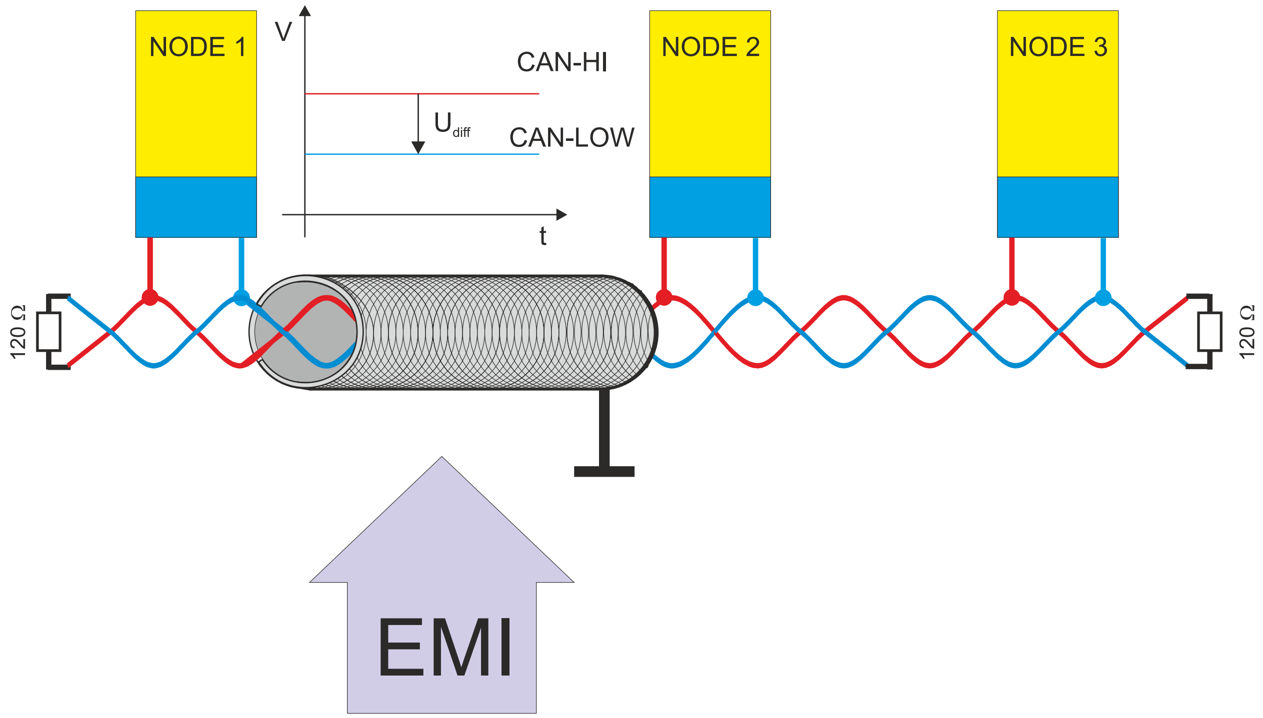
CAN is a 2-wire bus system, to which all participating devices are connected in parallel (i.e. using short drop lines). The bus must be terminated at each end with a 120 (or 121) Ohm terminating resistor to prevent reflections. This is also necessary even if the cable lengths are very short!




Since the CAN signals are represented on the bus as the difference between the two levels, the CAN leads are not very sensitive to incoming interference (EMI): Both leads are affected, so the interference has very little effect on the difference.




Additional shielding of the twisted wires can be used to further reduce EMI interference.




**Bus length**

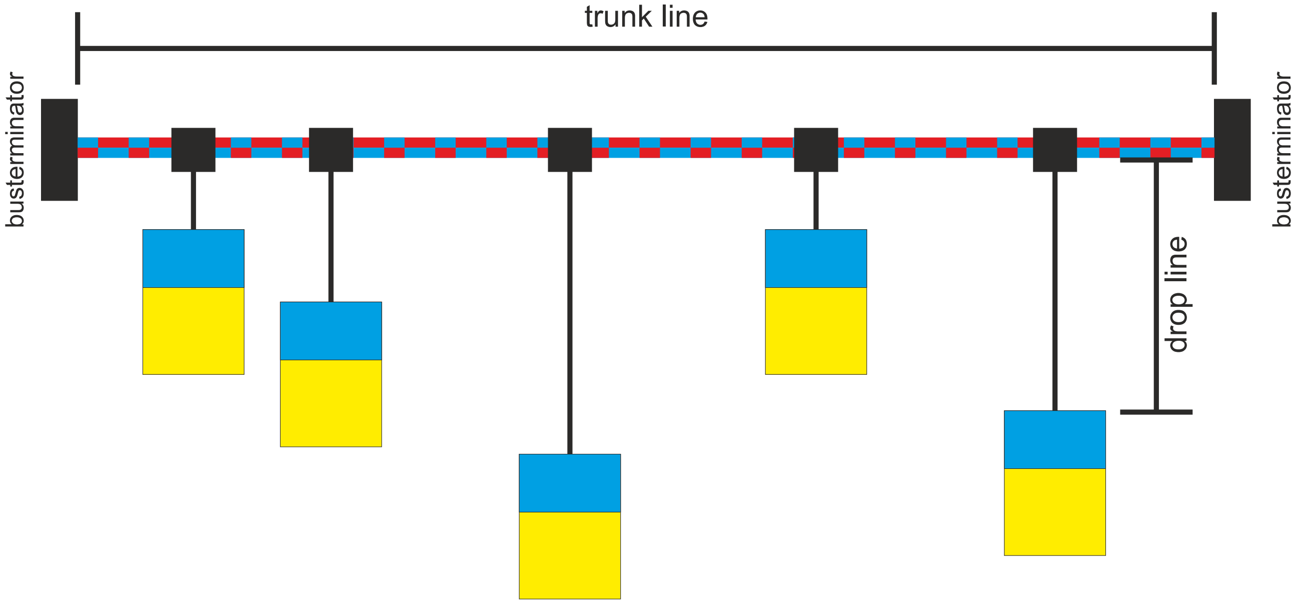
The maximum length of a CAN bus is primarily limited by the signal propagation delay. The multi-master bus access procedure (arbitration) requires signals to reach all the nodes at effectively the same time (before the sampling within a bit period). Since the signal propagation delays in the CAN connecting equipment (transceivers, opto-couplers, CAN controllers) are almost constant, the line length must be chosen in accordance with the baud rate:

| Baud rate | Bus length |
|---|---|
| 1 Mbit/s | < 20 m* |
| 500 kbit/s | < 100 m |
| 250 kbit/s | < 250 m |
| 125 kbit/s | < 500 m |
| 50 kbit/s | < 1000 m |
| 20 kbit/s | < 2500 m |
| 10 kbit/s | < 5000 m |

*) A figure of 40m at 1 Mbit/s is often found in the CAN literature. This does not, however, apply to networks with optically isolated CAN controllers. The worst case calculation for opto-couplers yields a figure 5 m at 1 Mbit/s - in practice, however, 20 m can be reached without difficulty.

It may be necessary to use repeaters for bus lengths greater than 1000 m.

**Drop lines**

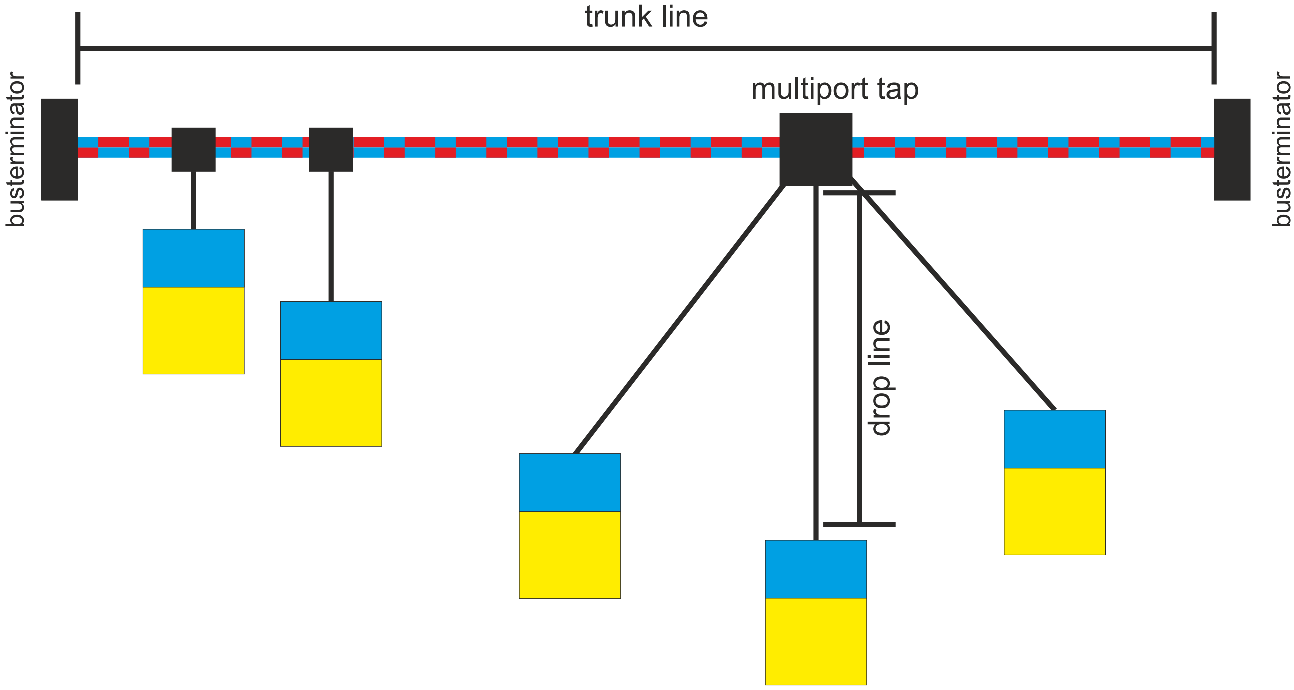



Drop lines must always be avoided as far as possible, since they inevitably cause reflections. The reflections caused by drop lines are not however usually critical, provided they have decayed fully before the sampling time. In the case of the bit timing settings selected in the Bus Couplers it can be assumed that this is the case, provided the following drop line lengths are not exceeded:

| Baud rate | Drop line length | Total length of all drop lines |
|---|---|---|
| 1 Mbit/s | < 1m | < 5 m |
| 500 kbit/s | < 5 m | < 25 m |
| 250 kbit/s | < 10 m | < 50 m |
| 125 kbit/s | < 20 m | < 100 m |
| 50 kbit/s | < 50 m | < 250 m |

Drop lines must not have terminating resistors.

**Star Hub (Multiport Tap)**



Shorter drop line lengths must be maintained when passive distributors ("multiport taps"), such as the BECKHOFF ZS5052-4500 Distributor Box. The following table indicates the maximum drop line lengths and the maximum length of the trunk line (without the drop lines):

| Baud rate | Drop line length with multiport topology | Trunk line length (without drop lines) |
| --- | --- | --- |
| 1 Mbit/s | < 0.3 m | < 25 m |
| 500 kbit/s | < 1.2 m | < 66 m |
| 250 kbit/s | < 2.4 m | < 120 m |
| 125 kbit/s | < 4.8 m | < 310 m |

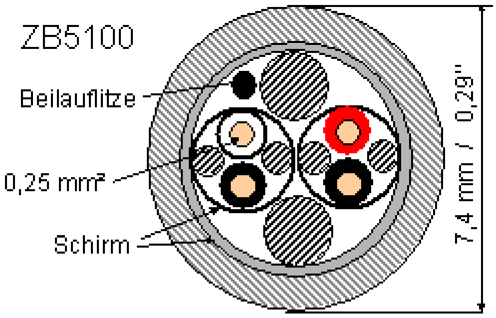
### CAN cable

Screened twisted-pair cables (2x2) with a characteristic impedance of between 108 and 132 Ohm is recommended for the CAN wiring. If the CAN transceiver's reference potential (CAN ground) is not to be connected, the second pair of conductors can be omitted. (This is only recommended for networks of small physical size with a common power supply for all the participating devices).

### ZB5100 CAN Cable

A high quality CAN cable with the following properties is included in BECKHOFF's range:

- 2 x 2 x 0.25 mm² (AWG 24) twisted pairs, cable colors: red/black + white/black
- double screened
- braided screen with filler strand (can be attached directly to pin 3 of the 5-pin connection terminal),
- flexible (minimum bending radius 35 mm when bent once, 70 mm for repeated bending)
- characteristic impedance (60kHz): 120 ohm
- conductor resistance < 80 Ohm/km
- sheath: grey PVC, outside diameter 7.3 +/- 0.4 mm
- Weight: 64 kg/km.
- printed with "BECKHOFF ZB5100 CAN-BUS 2x2x0.25" and meter marking (length data every 20cm)

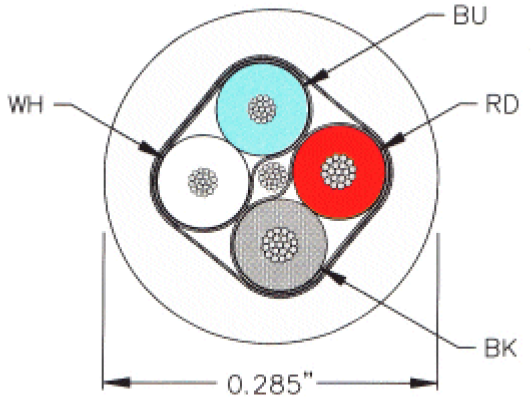



### ZB5200 CAN/DeviceNet Cable

The ZB5200 cable material corresponds to the DeviceNet specification, and is also suitable for CANopen systems. The ready-made ZK1052-xxxx-xxxx bus cables for the Fieldbus Box modules are made from this cable material. It has the following specification:

- 2 x 2 x 0.34 mm² (AWG 22) twisted pairs
- double screened braided screen with filler strand
- characteristic impedance (1 MHz): 126 ohm
- conductor resistance 54 Ohm/km
- sheath: grey PVC, outside diameter 7.3 mm
- printed with "InterlinkBT DeviceNet Type 572" as well as UL and CSA ratings
- stranded wire colors correspond to the DeviceNet specification
- UL recognized AWM Type 2476 rating

- CSA AWM I/II A/B 80°C 300V FT1
- corresponds to the DeviceNet "Thin Cable" specification




**Shielding**

The screen is to be connected over the entire length of the bus cable, and only galvanically grounded at one point, in order to avoid ground loops.
The design of the screening, in which HF interference is diverted through R/C elements to the mounting rail assumes that the rail is appropriately earthed and free from interference. If this is not the case, it is possible that HF interference will be transmitted from the mounting rail to the screen of the bus cable. In that case the screen should not be attached to the couplers - it should nevertheless still be fully connected through.

**Cable colors**

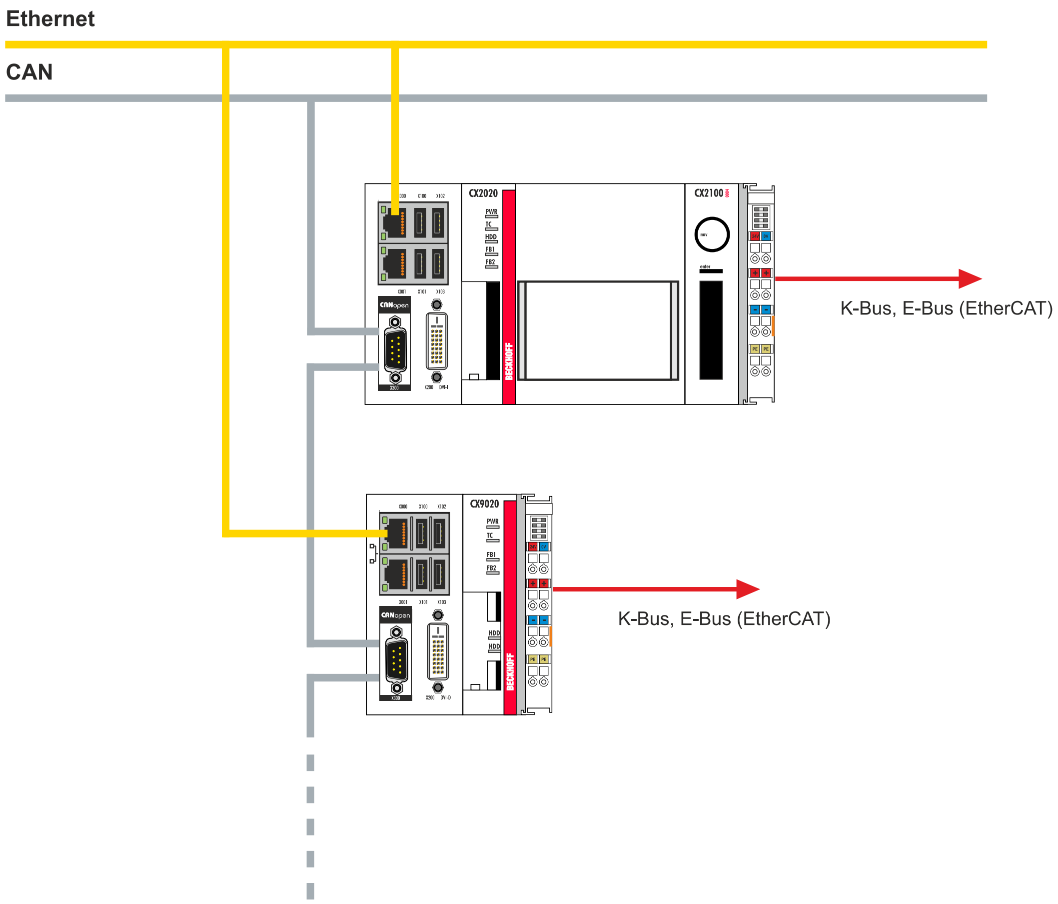
Recommended application of Beckhoff CAN cables:

| Function | ZB5100 cable color | ZB5200 cable color |
|---|---|---|
| CAN Ground | **black/** (red) | **black** |
| CAN Low | **black** | **blue** |
| Shield | Filler strand | Filler strand |
| CAN high | **white** | **white** |
| not used | **(red)** | **(red)** |

## 4.6 Topology

All devices are connected in parallel. The bus must be terminated at each end with a 120 ohm termination resistor. CANopen limits the number of devices per network to 64.

The maximum possible network size is limited by the data rate. For sample, at 1 Mbit/s a network size of 20 m is possible, at 50 kbit/s a network size of 1000 m. At lower data rates the network size can be increased with repeaters. Repeaters also enable the configuration of tree structures.



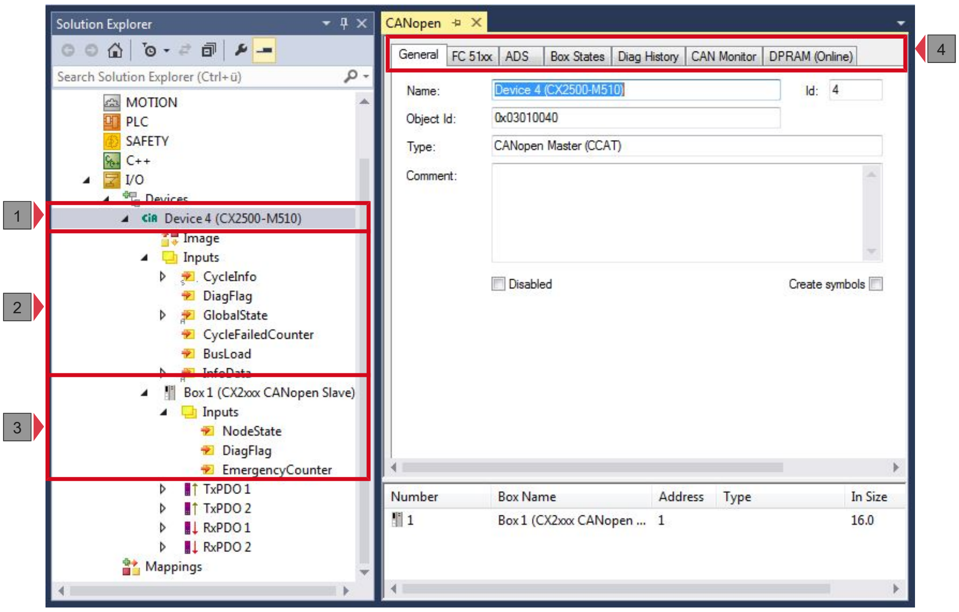
# 5 TwinCAT tabs

In TwinCAT, information and settings for the CANopen interface are added under tabs. The main TwinCAT tabs are described in this section. In addition, the section illustrates how the CANopen interface is displayed in the tree view under TwinCAT.

The tree view and the tabs for a CANopen interface are identical under TwinCAT2 and TwinCAT3.

## 5.1 Tree view

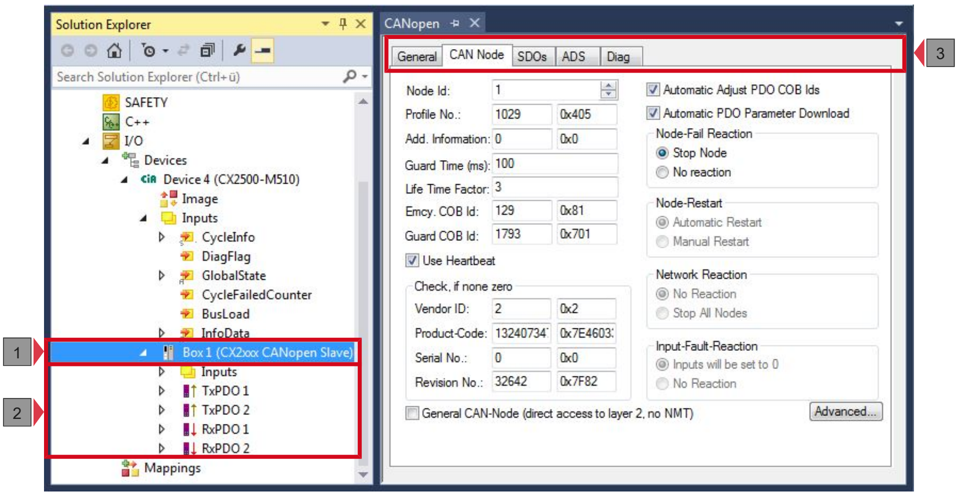
A CANopen master and a CANopen slave are displayed as follows in the tree view:




In this sample the slave was linked to the master. TwinCAT was then scanned for the master, and the master was added in TwinCAT together with the slave.

| No. | Description |
|---|---|
| 1 | The device name of the master is shown in brackets. All CANopen slaves are added under the master. |
| 2 | Under the CANopen master, status messages are listed as input variables. The variables can be linked with the PLC and used for diagnostic purposes (e.g. error codes, counters, etc.). |
| 3 | CANopen slaves are added under the master, labelled as box and numbered consecutively. The device name appears in brackets after it.<br>Each CANopen slave has its own input variables for diagnostic purposes, which indicate the state of the communication. |
| 4 | Further settings for the CANopen master or slave can be implemented under the tabs.<br>Other tabs are displayed, depending on whether the master or slave is selected in the tree view. |

A CANopen slave and the corresponding tabs are shown as follows in the tree view:




| No. | Description |
|---|---|
| 1 | Under the CANopen slave, status messages are listed as input variables. The variables can be linked with the PLC and used for diagnostic purposes. |
| 2 | The process data objects (PDO) are displayed under the CANopen slave. At this point the variables for the data transfer are also created. The variables can be linked with the PLC.<br>The data transfer direction is described from the perspective of the slave:<br>• RxPDOs are received by the device.<br>• TxPDOs are sent by the device. |
| 3 | Further settings for the CANopen slave can be implemented under the tabs.<br>Other tabs are displayed, depending on whether slave or other entries are selected in the tree view. |

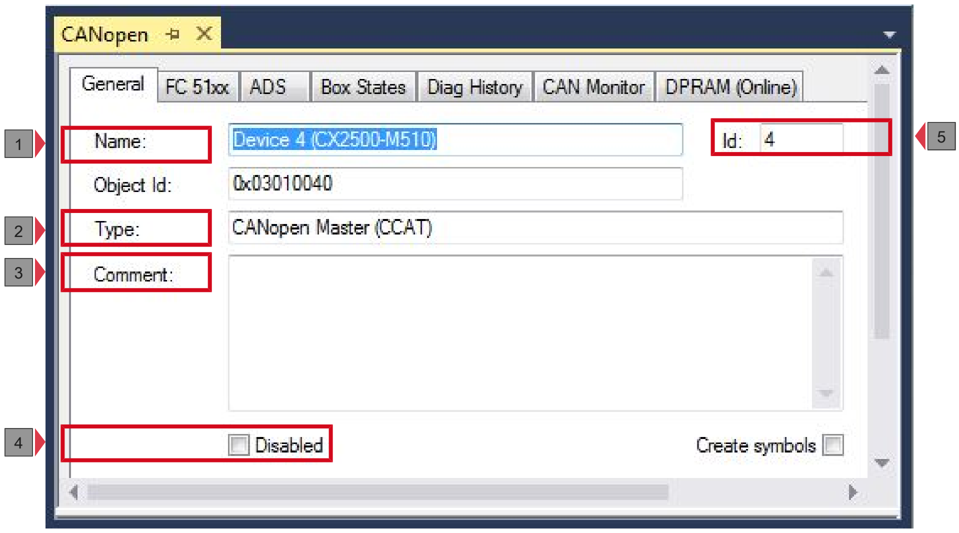
When the PLC process image is read, the variables for status messages and the variables under the process data objects can be linked with the variables from the PLC program. Double-click on a variable name in the tree view to open the link dialog. The link variables are identified with a small arrow icon.

Further information about TwinCAT can be found in the TwinCAT documentation on the Beckhoff website: www.beckhoff.com

## 5.2 CANopen master

### 5.2.1 General

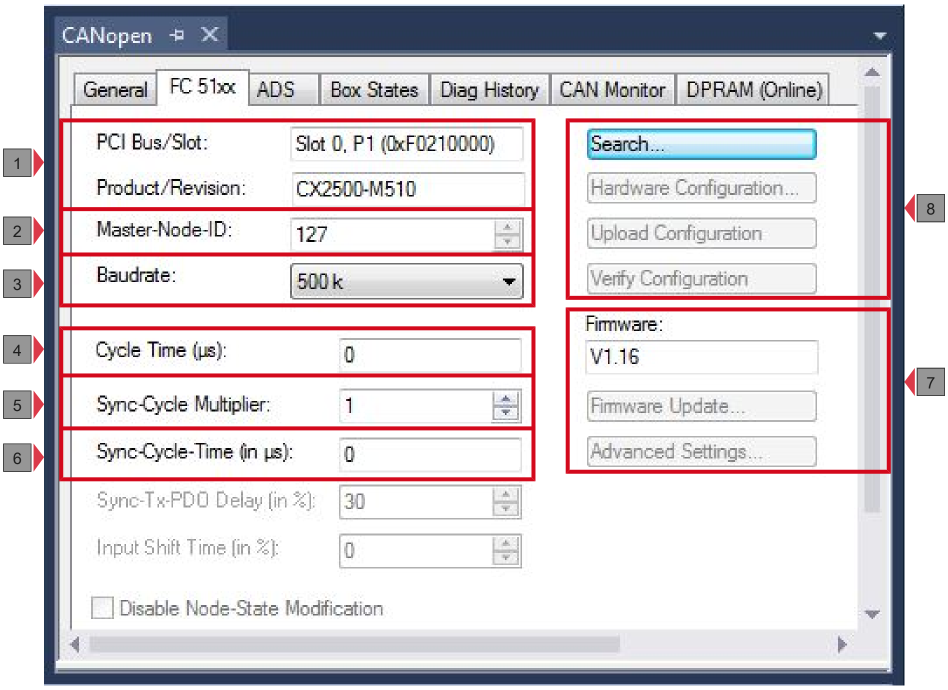
The General tab contains general information for a CANopen device, including name, type and ID.




| No. | Description |
|---|---|
| 1 | Name of the CANopen device |
| 2 | CANopen device type |
| 3 | Here you can add a comment (e.g. notes relating to the system component) |
| 4 | Here you can disable the CANopen device |
| 5 | Running No. |

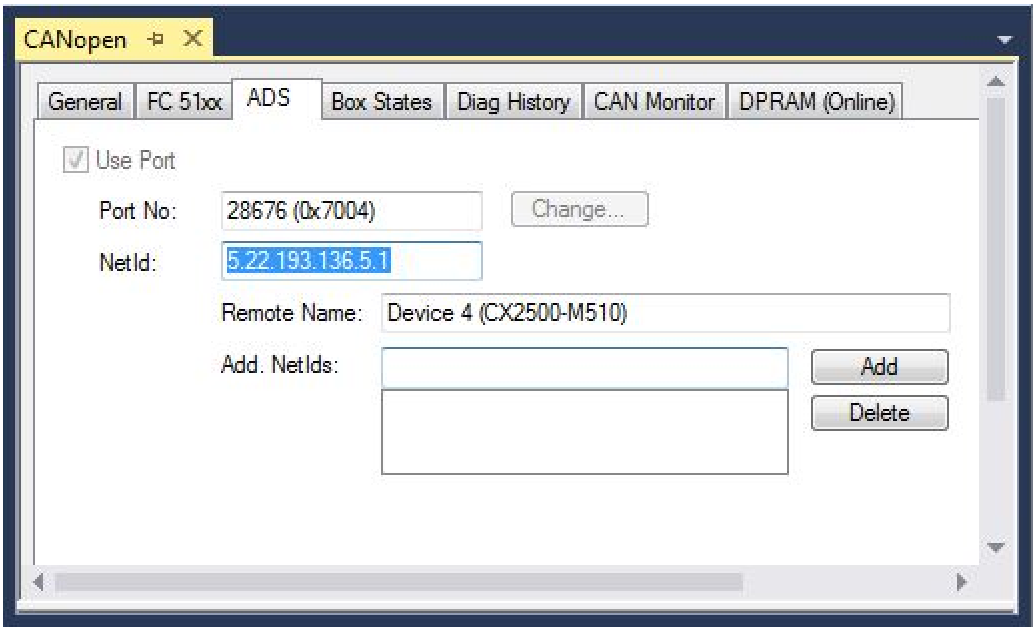
The CANopen device can be switched off via this tab. A comment field offers the option to add a label, in order to provide additional information on the device.

## 5.2.2 FC 51xx




| No. | Description |
|---|---|
| 1 | Name of the physical interface.<br>Name and type of the CANopen device. |
| 2 | Name of the CANopen master. Range between 1 and 127. Determines the identifier of the master heartbeat telegram. Ensure that it is not the same as a slave node address. |
| 3 | The baud rate is set here. Automatically tests whether the connected slave supports this baudrate. |
| 4 | Displays the cycle time of the corresponding highest priority task. |
| 5 | With CANopen it is often the case that event-driven communication is combined with cyclic synchronous communication. In order to be able to respond to events quickly, the task cycle time must be less than the cycle time of the sync telegram.<br>If the sync cycle multiplier is set to values > 1, the TwinCAT task is called repeatedly before the sync telegram is sent again. |
| 6 | The cycle time of the sync telegram is displayed here. It is determined by the cycle time of the highest priority task, its process data and the sync cycle multiplier.<br>Sync cycle time = cycle time **x** sync cycle multiplier |
| 7 | The current firmware version is displayed here. |
| 8 | The **Search** button is used to find and select the required physical interface, if not done automatically already. |

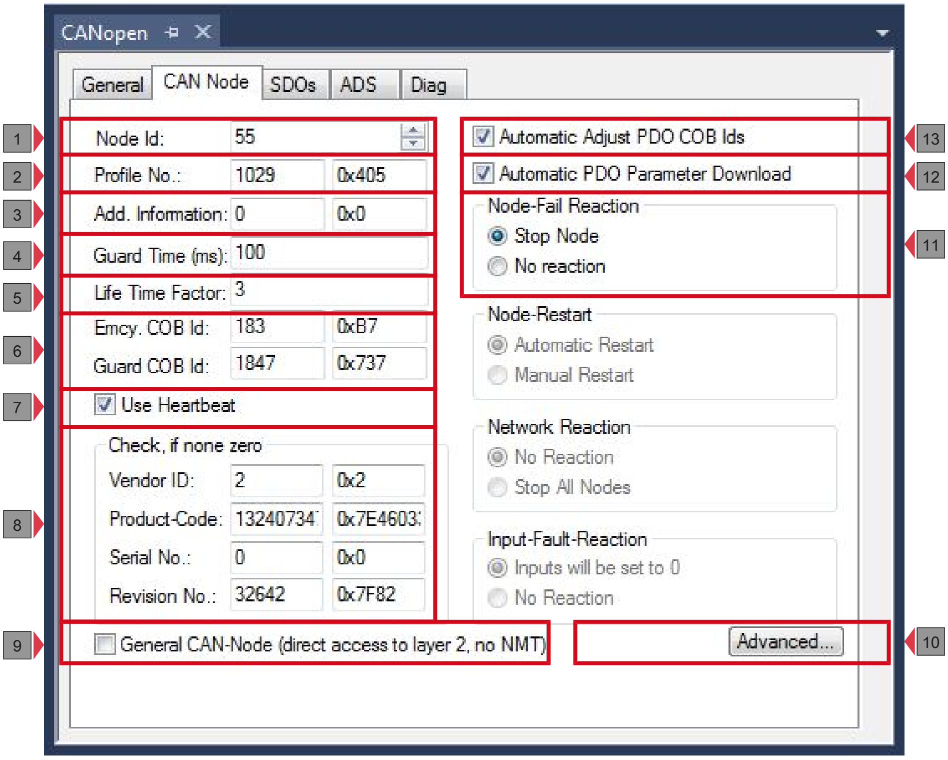
## 5.2.3 ADS




The CANopen master is an ADS device with its own Net ID, which can be modified here. All ADS services (diagnostics, acyclic communication) sent to the CANopen master must use this Net ID and port no.

## 5.3 CANopen slave

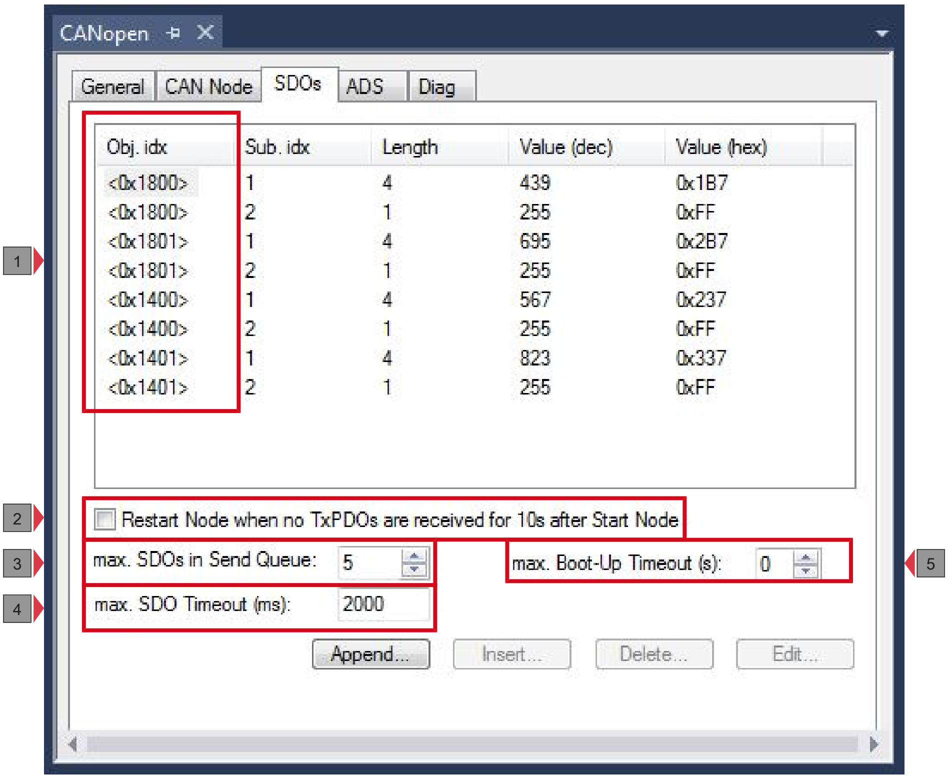
### 5.3.1 CAN node



| No. | Description |
|---|---|
| 1 | The address is set here. |
| 2 | After CANopen, the parameter 0x1000 "Device Type" contains the number of the supported device profile in both the lowest value bytes. This number is entered here and compared with the parameter in the device on system startup. If no device profile is supported, the parameter will contain the value 0. |
| 3 | Add. information:<br>Add. information is contained in the two most significant bytes of the object directory entry 0x1000 (device type). A set/actual configuration comparison is only made if the profile no. or add. information (i.e. object directory entry 0x1000) is set to a value other than zero. If the expected values do not match the actual values on system startup, the node start is aborted and a corresponding error message is displayed on the Diag tab. |
| 4 | Guard time:<br>The guard time determines the interval in which the node is monitored (node guarding). The entered value is rounded up to the next the multiple of 10 ms. 0 signifies no monitoring. |
| 5 | Life time factor:<br>Guard time x life time factor determines the watchdog length for the mutual monitoring of master and slave. The entry 0 means that the slave does not monitor the master. If 0 is entered, the master directly takes the guard time as watchdog length.<br>The heartbeat protocol is also supported, and the system initially tries to initiate this form of node monitoring on the CANopen node. If this attempt fails, guarding is activated. |
| 6 | Emcy COB ID / Guard COB ID are identifiers for emergency messages or the guarding protocol. They result from the node address. |
| 7 | Heartbeat is used for monitoring of the node. If heartbeat is disabled, guarding is used for monitoring.<br>The guard time as producer heartbeat time and (guard time x lifetime factor) as consumer heartbeat time are entered. In this case a heartbeat telegram with the smallest configured guard time sent. The guard time can be set individually for each node. |
| 8 | If values other than zero are entered here, these identity object inputs (0x1018 in the object directory) are read off at the system StartUp and compared with the configured values. The corresponding node will be started only if the values coincide. It is also possible to compare only some of the values (e.g. the vendor ID and the product code). In this case, parameters that are not used must be set to zero. |
| 9 | If this option is selected, the whole CANopen network management is disabled for this device, i.e. it is not started and monitored etc. The PDO entries are regarded as pure CAN telegrams (layer 2) and made available to the controller on an event-driven basis. |
| 10 | Opens a window with further settings, which can be enabled:<br>• Switch off upload object 0x1000.<br>• Switch off download object 0x1006.<br>• Switch off automatic sending of start node (then has to be sent manually).<br>• Continue to send start SDOs, in the event of a termination. |
| 11 | The option StopNode is used to set the node to "stopped" state after a fault. It can be used to set nodes to a safe state, although they can no longer be addressed via SDO. |
| 12 | If the option is selected, entries are created automatically in TwinCAT, which are transferred via SDO on system startup (see: SDOs [▶ 42] tab). |
| 13 | If the option is selected, the default identifiers of the process data objects are automatically adjusted if the node ID changes (see: no. 6). |

## 5.3.2 SDOs

The SDO tab is used to display/manage entries, which are sent to the node on startup.




| No. | Description |
|---|---|
| 1 | Object index entries in angle brackets were created automatically based on the current configuration.<br>Further entries can be created and managed via "Append", "Insert", Delete" and "Edit". |
| 2 | If this option is selected, the slave is restarted if no TxPDO was received after 10 seconds. |
| 3 | This option can be used to set the maximum number of SDOs in the send queue. |
| 4 | The maximum timeout (ms) for the SDO is set here. |
| 5 | The boot-up timeout (s) is set here. |

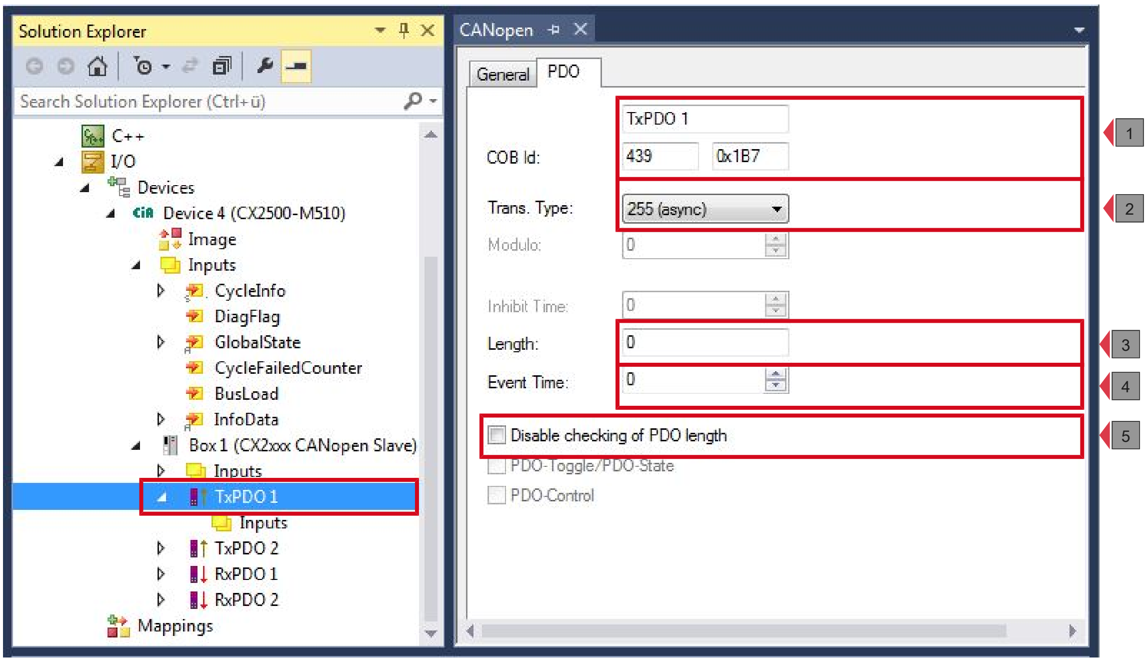
## 5.3.3 PDO

This tab appears if you click on a process data object (PDO) in the tree view.

Process Data Objects (PDOs) are CAN telegrams which transport process data without a protocol overhead.

- RxPDOs are received by the device.
- TxPDOs are sent by the device.

A device sends its input data with TxPDOs and receives the output data in the RxPDOs. This designation is retained in TwinCAT.




| No. | Description |
|---|---|
| 1 | CAN identifier of the PDO. For two send and receive PDOs per node, CANopen provides Default Identifiers. These can then be changed. |
| 2 | The Transmission Type determines the send behavior of the PDO. 255 corresponds to the event-driven sending (see: Setting the transmission type [▶ 73]). |
| 3 | The length of the PDO depends on the created variables and can therefore not be edited here. |
| 4 | Enter the value for the Event Timer in ms. For send PDOs (RxPDOs), PDOs are sent again after a timer has elapsed. For receive PDOs (TxPDOs), the arrived PDOs are monitored, and the box state of the node may be modified.<br>TwinCAT creates corresponding inputs in the node object directory on the basis of the parameters entered here. These are transferred via SDO at the system start. The entries can be viewed in the SDO tab (see: SDOs [▶ 42]).<br>This function can be deactivated via the checkbox **Automatic PDO Parameter Download** on the **CAN node** tab (see: CAN node [▶ 40]). |
| 5 | The PDO length check can be disabled here. |

# 6 Parameterization and commissioning

This documentation uses CANopen devices to illustrate the commissioning procedure. The configuration options shown in this section can be used for all Embedded PCs with CANopen interface.

The following devices are used in this documentation:

- CX2020-M510 (Embedded PC with optional CANopen master interface, D-sub socket, 9-pin)
- CX2500-B510 (Embedded PC with fieldbus module CX2500-B510 CANopen slave, D-sub socket, 9-pin)
- BK5100 (CANopen slave, D-sub socket, 9-pin)

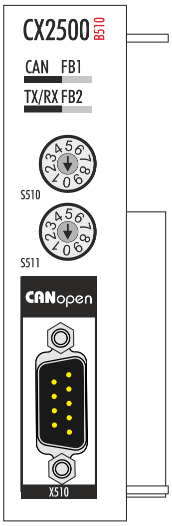
The TwinCAT 2 or TwinCAT 3 software is used for configuring the devices.

For further information see the TwinCAT 2 and TwinCAT 3 documentation, which is available from the Beckhoff website:
www.beckhoff.de

## 6.1 DIP switch

The CANopen fieldbus modules CX2500-M510 and CX2500-B510 feature two 10-pin address selection switches. The address selection switches can be used to set the CANopen address.

The address selection switch has no purpose in the master.




**Fieldbus module**

**Settings:**

- S510 for address x 1
- S511 for address x 10

Sample: If the switches are set to S510 = 2 and S511 = 1, the resulting address is = 12.

To ensure that the address set via the two address selection switches is used, the corresponding setting must be activated in TwinCAT (e.g. in TwinCAT 3 see: Setting the address with a DIP switch [▶ 69]).

## 6.2 PDO Parameterization

Even though the majority of CANopen networks operate satisfactorily with the default settings, i.e. with the minimum of configuration effort, it is wise at least to check whether the existing bus loading is reasonable: 80% bus loading may be acceptable for a network operating purely in cyclic synchronous modes, but for a network with event-driven traffic this value would generally be too high, as there is hardly any bandwidth available for additional events.

**Consider the Requirements of the Application**

The communication of the process data must be optimized in the light of application requirements which are likely to be to some extent in conflict. These include

- Little work on parameterization - useable default values are optimal
- Guaranteed reaction time for specific events
- Cycle time for regulation processes over the bus
- Safety reserves for bus malfunctions (enough bandwidth for the repetition of messages)
- Maximum baud rate - depends on the maximum bus length
- Desired communication paths - who is speaking with whom

The determining factor often turns out to be the available bus bandwidth (bus load).

**Determine the Baud Rate**

We generally begin by choosing the highest baud rate that the bus will permit. It should be borne in mind that serial bus systems are fundamentally more sensitive to interference as the baud rate is increased. The following rule therefore applies: just as fast as necessary. 1000 kbit/s are not usually necessary, and only to be unreservedly recommended on networks within a control cabinet where there is no electrical isolation between the bus nodes. Experience also tends to show that estimates of the length of bus cable laid are often over-optimistic - the length actually laid tends to be longer.

**Determine the Communication Type**

Once the baud rate has been chosen it is appropriate to specify the PDO communication type(s). These have different advantages and disadvantages:

- Cyclic synchronous communication provides an accurately predictable bus loading, and therefore a defined time behavior - you could say that the standard case is the worst case. It is easy to configure: The SYNC rate parameter sets the bus loading globally. The process images are synchronized: Inputs are read at the same time, output data is set valid simultaneously, although the quality of the synchronization depends on the implementation.

  The guaranteed reaction time under cyclic synchronous communication is always at least as long as the cycle time. The bus bandwidth is not exploited optimally, since old data, i.e. data that has not changed, is continuously transmitted. It is however possible to optimize the network through the selection of different SYNC multiples (transmission types 1...240), so that data that changes slowly is transmitted less often than, for instance, time-critical inputs. It must, however, be borne in mind that input states that last for a time that is shorter than the cycle time will not necessarily be communicated. If it is necessary for such conditions to be registered, the associated PDOs for asynchronous communication should be provided.
- Event-driven asynchronous communication is optimal from the point of view of reaction time and the exploitation of bus bandwidth - it can be described as "pure CAN". Your choice must, however, also take account of the fact that it is not impossible for a large number of events to occur simultaneously, leading to corresponding delays before a PDO with a relatively low priority can be sent. Proper network planning therefore necessitates a worst-case analysis. Through the use of, for instance, inhibit time, it is also necessary to prevent a constantly changing input with a high PDO priority from blocking the bus (technically known as a "babbling idiot"). It is for this reason that event driving is switched off by default in the device profile of analog inputs, and must be turned on specifically. Time windows for the transmit PDOs can be set using progress timers: the telegram is not sent again before the inhibit time has elapsed, and not later than the time required for the progress timer to complete.

- The communication type is parameterized by means of the transmission type (see: Setting the transmission type [▸ 73]).

It is also possible to combine the two PDO principles. It can, for instance, be helpful to exchange the set and actual values of an axis controller synchronously, while limit switches, or motor temperatures with limit values are monitored with event-driven PDOs. This combines the advantages of the two principles: synchronicity for the axis communication and short reaction times for limit switches. In spite of being event-driven, the distributed limit value monitoring avoids a constant addition to the bus load from the analog temperature value.

In this example it can also be of value to deliberately manipulate the identifier allocation, in order to optimize bus access by means of priority allocation: the highest priority is given to the PDO with the limit switch data, and the lowest to that with the temperature values.

Optimization of bus access latency time through modification of the identifier allocation is not, however, normally required. On the other hand the identifiers must be altered if masterless communication is to be made possible (PDO linking). In this example it would be possible for one RxPDO for each axis to be allocated the same identifier as the limit switch TxPDO, so that alterations of the input value can be received without delay.

**Determining the Bus Loading**

It is always worth determining the bus loading. But what bus loading values are permitted, or indeed sensible? It is first necessary to distinguish a short burst of telegrams in which a number of CAN messages follow one another immediately - a temporary 100% bus loading. This is only a problem if the sequence of receive interrupts that it caused at the CAN nodes can not be handled. This would constitute a data overflow (or CAN queue overrun). This can occur at very high baud rates (> 500 kbit/s) at nodes with software telegram filtering and relatively slow or heavily loaded microcontrollers if, for instance, a series of remote frames (which do not contain data bytes, and are therefore very short) follow each other closely on the bus (at 1 Mbit/s this can generate an interrupt every 40 µs; for example, an NMT master might transmit all its guarding requests in an unbroken sequence). This can be avoided through skilled implementation, and the user should be able to assume that the device suppliers have taken the necessary trouble. A burst condition is entirely normal immediately after the SYNC telegram, for instance: triggered by the SYNC, all the nodes that are operating synchronously try to send their data at almost the same time. A large number of arbitration processes take place, and the telegrams are sorted in order of priority for transmission on the bus. This is not usually critical, since these telegrams do contain some data bytes, and the telegrams trigger a sequence of receive interrupts at the CAN nodes which is indeed rapid, but is nevertheless manageable.

Bus loading most often refers to the value averaged over several primary cycles, that is the mean value over 100-500 ms. CAN, and therefore CANopen, is indeed capable of managing a bus loading of close to 100% over long periods, but this implies that no bandwidth is available for any repetitions that may be necessitated by interference, for asynchronous error messages, parameterization and so on. Clearly, the dominant type of communication will have a large influence on the appropriate level of bus loading: a network with entirely cyclic synchronous operation is always in any case near to the worst case state, and can therefore be operated with values in the 70-80% range. The figure is very hard to state for an entirely event-driven network: an estimate must be made of how many events additional to the current state of the system might occur, and of how long the resulting burst might last - in other words, for how long the lowest priority message will be delayed. If this value is acceptable to the application, then the current bus loading is acceptable. As a rule of thumb it can usually be assumed that an event-driven network running with a base loading of 30-40% has enough reserve for worst-case scenarios, but this assumption does not obviate the need for a careful analysis if delays could have critical results for the plant.

The BECKHOFF CANopen-Master devices indicate the bus loading via the System Manager. This variable can also be processed in the PLC, or can be displayed in the visualization system.

The amount data in the process data objects is of course as relevant as the communication parameters: the PDO mapping.

## 6.3 Parameterization with TwinCAT 2

This section illustrates how CANopen devices can be parameterized with the aid of TwinCAT 2. A total of three devices are used for the sample, including a CANopen master, to which two CANopen slaves are connected.

First, the process of finding and selecting a target system in TwinCAT is illustrated. Next, a CANopen slave is added and parameterized in TwinCAT, and the CANopen address of the slave is set. Then a PLC project is created and added in TwinCAT. Then, the variables from the PLC project are linked with the hardware, and the finished configuration is loaded on the CANopen slave.

In the last step, the CANopen master is added in TwinCAT, and the two CANopen slaves are located via the master.

### 6.3.1 Searching for target systems

Before you can work with the devices, you must connect your local computer to the target device. Then you can search for the devices with the help of the IP address or the host name.

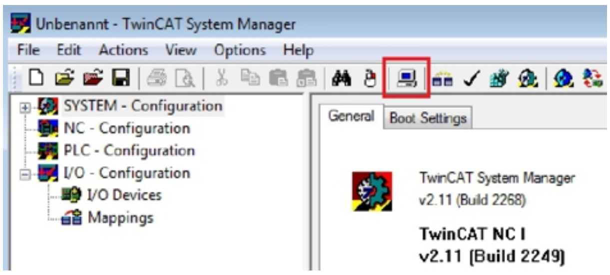
The local PC and the target devices must be connected to the same network or directly to each other via an Ethernet cable. In TwinCAT a search can be performed for all devices in this way and project planning subsequently carried out.

Prerequisites for this step:

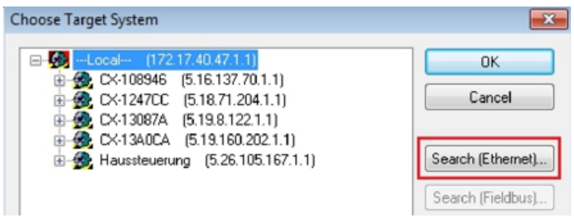
- TwinCAT 2 must be in Config mode.
- IP address or host name of the device. The host name is composed of CX- and the last 3 bytes of the MAC address. The MAC address is located on the side of the device.

**Search for the devices as follows:**

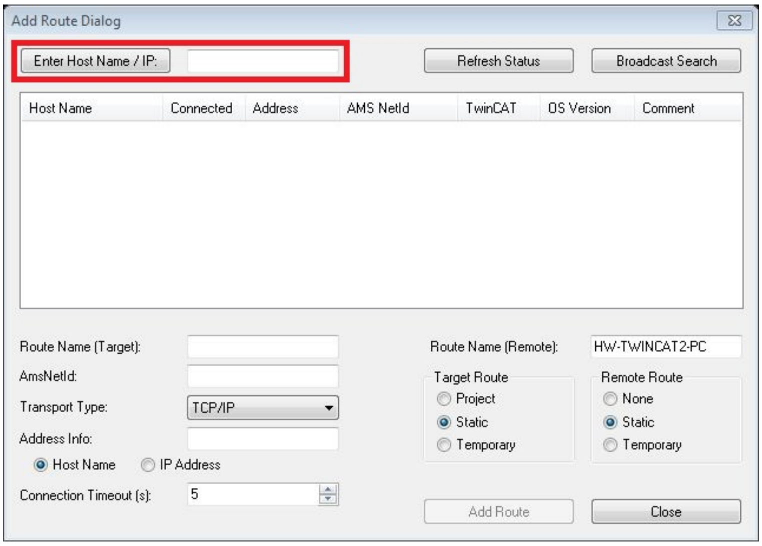
1. Click on **File > New** in the menu at the top.
2. Click on **Choose Target System** in the toolbar at the top.



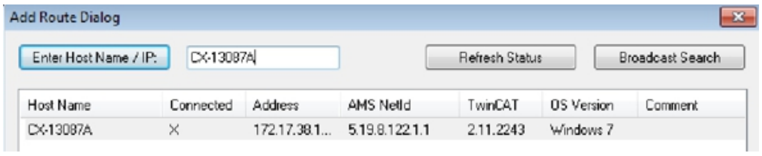

3. Click on **Search (Ethernet)**.



4. Type the host name or the IP address of the device into the **Enter Host Name / IP** box and press **[Enter]**.




5. Mark the device found and click on **Add Route**.

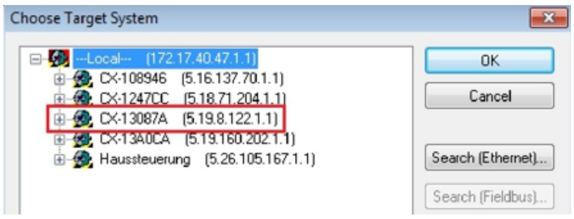



The Logon Information window appears.

6. Enter the user name and password for the CX in the **User Name** and **Password** fields and click **OK**.


The following information is set as standard in CX devices:
**User name:** Administrator **Password:** 1

7. If you do not wish to search for any further devices, click on **Close** to close the Add Route Dialog. The new device is displayed in the Choose Target System window.
8. Mark the device that you wish to set as the target system and click on **OK**.




⇨ You have successfully searched for a device in TwinCAT and inserted the device as the target system. The new target system is displayed in the bottom right-hand corner together with the host name and IP address (AMS Net ID).


Using this procedure you can search for all available devices and also switch between the target systems at any time. Next, you can append the device to the tree view in TwinCAT.

## 6.3.2 Add CANopen slave

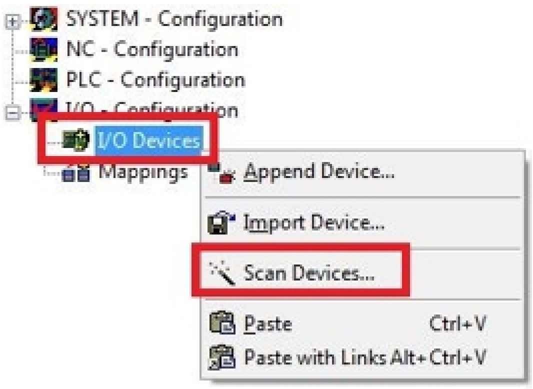
The sample shows a CX2020 CANopen slave with CX2500-B510 fieldbus module, connected to the CANopen master. In order to ensure that the CANopen slave is configured and subsequently detected by the CANopen master with all inputs and outputs, the CANopen slave first must be added in TwinCAT.

Prerequisites for this step:

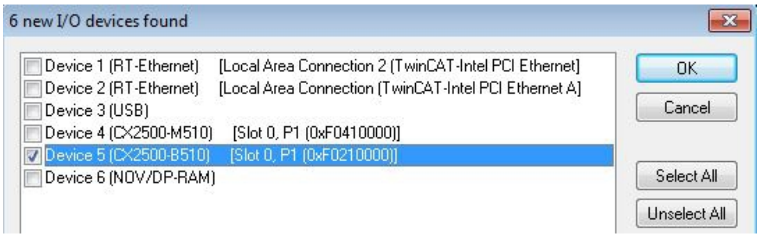
- A scanned and selected target device with CANopen slave. This sample uses a CX2020 with CX2500-B510 fieldbus module.

**Add the CANopen slave as follows:**

1. In the tree view on the left, right-click on **I/O Devices**.
2. In the context menu click on **Scan Devices**.

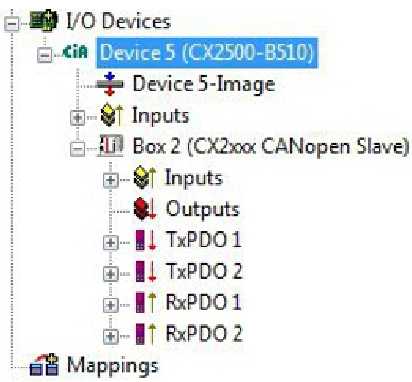



3. Select the devices you want to use and confirm the selection with **OK**.




4. Confirm the request with Yes, in order to look for boxes.

⇨ The CANopen slave was successfully added in TwinCAT 2 and is displayed in the tree view with the inputs and outputs.




In the next step you can extend the process image by creating additional virtual slaves. Or you can set the address, once the slave configuration is complete.

## 6.3.3 Creating a virtual slave

Additional virtual slaves can be created on the same hardware interface. This enables more data to be exchanged with a CANopen master, or a connection with a second CANopen master can be established.

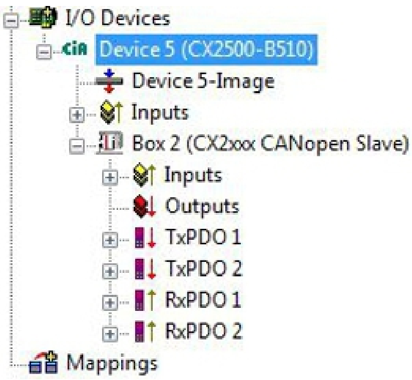
Each virtual slave is assigned a dedicated address via TwinCAT and is configured like an independent device for the CANopen master.

Prerequisites for this step:

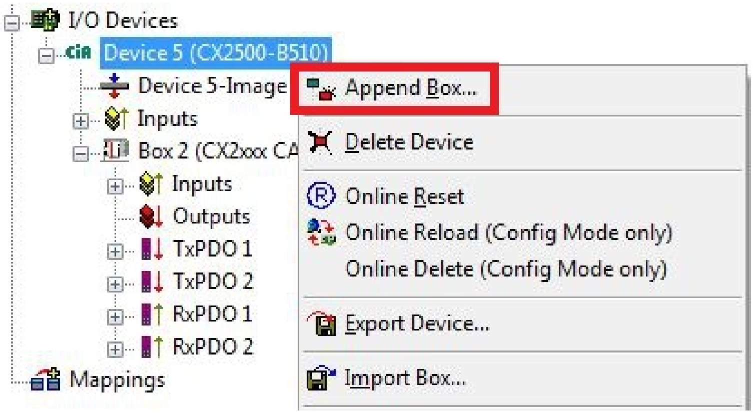
- A CANopen slave, created in TwinCAT.

**Create a virtual slave as follows:**

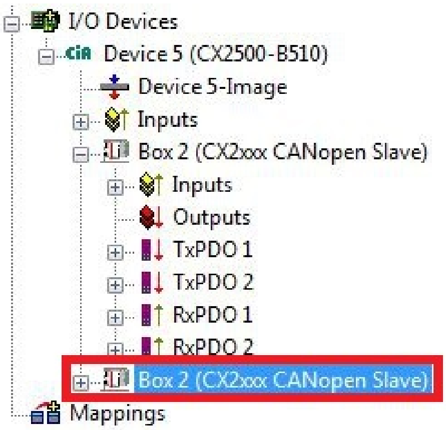
1. Right-click on a CANopen slave in the tree view on the left.




2. Click on **Append Box** in the context menu.




⇨ A further box (virtual slave) is created.




Variables for the virtual slave can now be created. In the next step you can set the address for the slave.

## 6.3.4 Setting the address

Once the CANopen slave was successfully added in TwinCAT, the address of the CANopen slave can be set. Devices with a DIP switch have a preset address. The address on the DIP switch must match the address set in TwinCAT.

For devices without DIP switch the address is only set in TwinCAT.

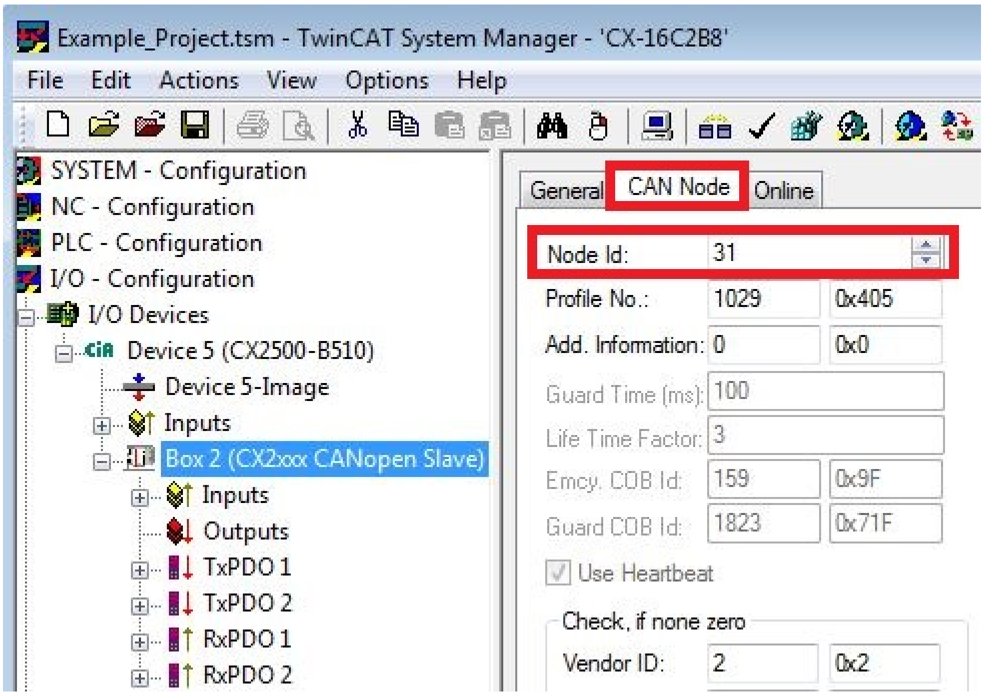
In this step the address is set in TwinCAT, so that the CANopen slave can be reached by the CANopen master via this address.

Prerequisites for this step:

- An added CANopen slave in TwinCAT.

**Parameterize the CANopen slave as follows:**

1. Click on a slave box.
2. Click on the **CAN Node** tab.
3. Enter a value for the CANopen address in the **Node Id** field, e.g. “31”.




⇨ You have set the address successfully. The CANopen master can reach the CANopen slave with the set address.
You can now create further PDOs.

## 6.3.5 Setting the address with a DIP switch

To ensure that the address set via the address selection switch is used, the corresponding setting must be enabled in TwinCAT.

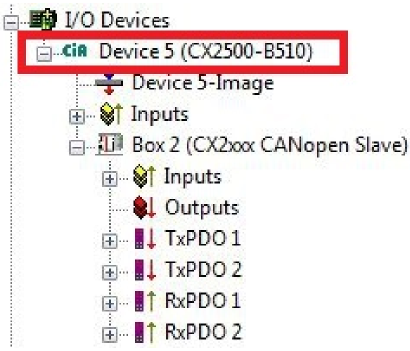
The corresponding setting in TwinCAT is only active, if the CANopen device has an address selection switch.

Prerequisites for this step:

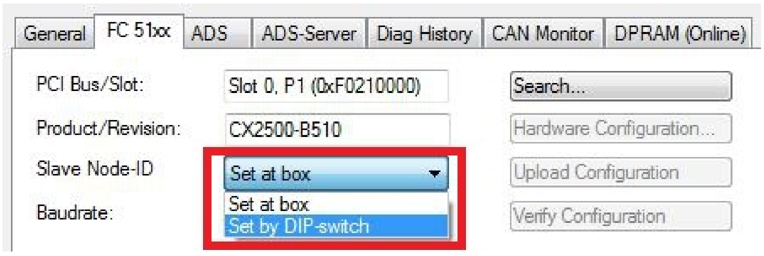
- CANopen device with address selection switch.
- CANopen device added in TwinCAT.

**Set the slave node id as follows:**

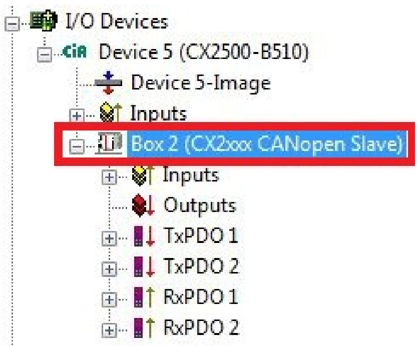
1. Click on the CANopen device in the tree view on the left.



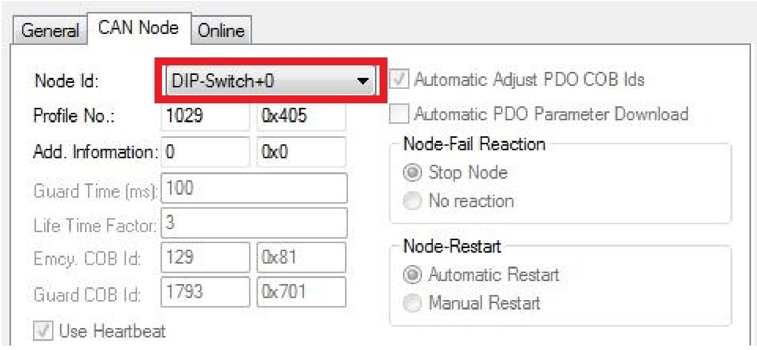

2. Click on theFC 51xxtab and select the option **Set by DIP-switch**under **Slave Node-ID**.




3. Click on the CANopen box in the tree view on the left.



4. Click on the **CAN Node** tab and select the option **DIP-Switch+0** under **Node Id**.
In this example the number 0 is added to the address set on the DIP switch. Other numbers (0-126), e.g. for virtual slaves, may also be added to the address.




⇨ You have set the slave node id successfully, and the CANopen slave will use the set address from now on. You can now create further PDOs.

## 6.3.6 Creating further PDOs

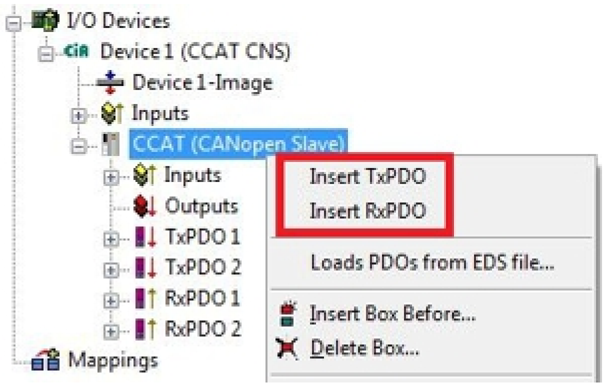
The CANopen slave can exchange up to 16 PDOs (each with 8 bytes of process data) with the CANopen master in input and output direction. By default 2 PDOs are created in Tx and Rx direction.
Here we shown how to create further PDOs for a CANopen slave.

Prerequisites for this step:

- A CANopen slave added in the tree view.

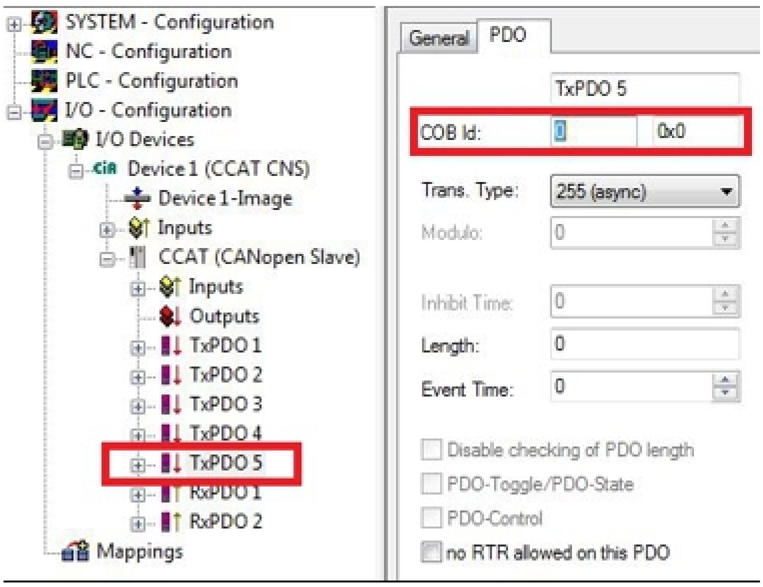
**Create the PDOs as follows:**

1. Right-click on a CANopen slave in the tree view.
2. In the context menu, click on **Insert TxPDO** or **Insert RxPDO** in order to create PDOs in Tx or Rx direction.




The new TxPDOs or RxPDOs are added in the tree view below the already created PDOs and numbered consecutively.

*Notice* From the fifth PDO in Tx or Rx direction the COB ID is no longer entered automatically (see the following diagram).




3. From the fifth PDO in Tx or Rx direction click on the **PDO** tab.
4. Enter the required value in the **COB ID** field.

⇨ You have successfully created further PDOs; in the next step you can create variables for the data exchange under the PDOs.

## 6.3.7 Creating variables

In TwinCAT the PDOs are filled with variables, which can later be linked with the PLC program. This section describes how to create variables.

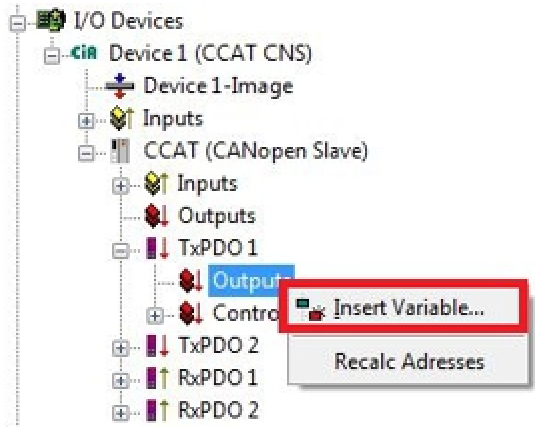
Prerequisites for this step:

- Newly created PDOs, which are to be filled with variables.

**Create the variables as follows:**

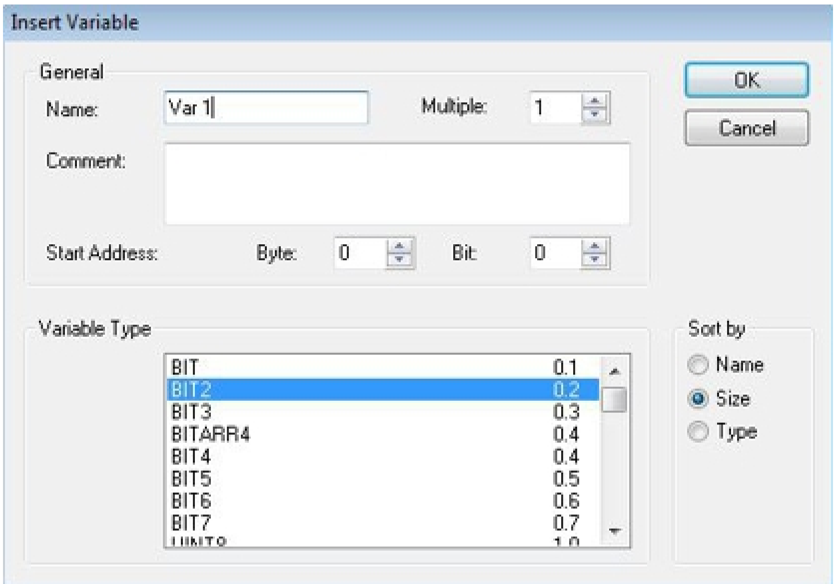
1. In the tree view click on a TxPDO or RxPDO to show more information.

2. Right-click on Outputs or Inputs, depending on whether a TxPDO or RxPDO is selected.




3. In the context menu click on **Insert Variable**.
   The **Insert Variable** window appears.

4. Click on the required variable, then click **OK**.




⇨ You have successfully created a variable. The new variable is shown in the tree view on the left. In this way you can add further variables for the CANopen slave.
In the next step you can specify the transmission type, thereby specifying how the process data objects are transferred.

## 6.3.8 Setting the transmission type

The transmission type determines how the process data objects are transferred. The transmission type for the RxPDOs and TxPDOs is set on the PDO tab.

The available transmission types are: acyclic synchronous, cyclic synchronous, and asynchronous.

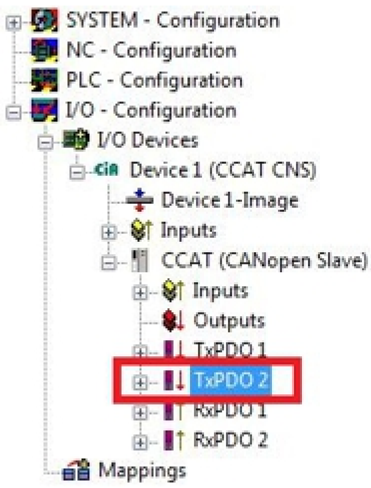
| Transmission type: | Acyclic Synchronous | Cyclic Synchronous | Asynchronous |
|---|---|---|---|
| Name in TwinCAT: | (acyc, sync) | (cyc, sync) | (async) |

Prerequisites for this step:

- A CANopen slave with process data objects (PDO) added in TwinCAT

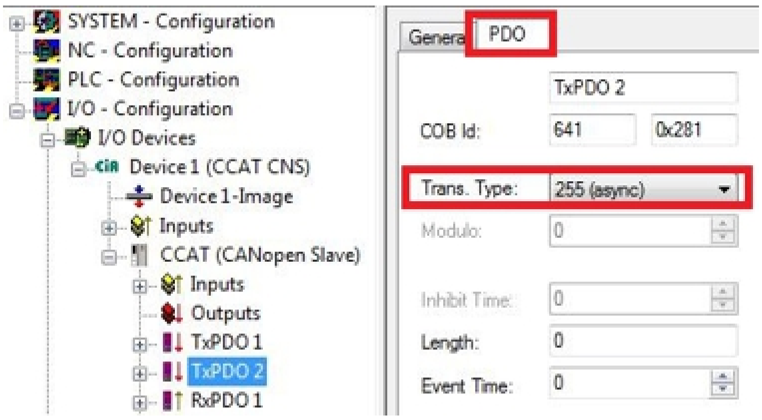
**Specify the transmission type as follows:**

1. In the tree view, left-click on a process data object (PDO).




2. Click on the **PDO** tab.

3. Select the required transmission type under **Trans. Type**.




⇨ You have successfully specified a transmission type for a process data object. The transmission types for the remaining process data objects are specified in the same way.
Next, you can create a PLC project for the CANopen slave.

### 6.3.9 Creating a PLC project

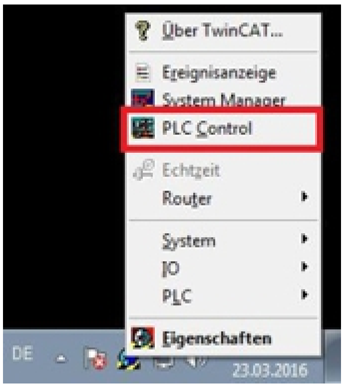
Use PLC Control to create a PLC project. The next steps describe how to create a PLC project in TwinCAT and add it in the tree view.

Prerequisites for this step:

- An Embedded PC, added in TwinCAT.

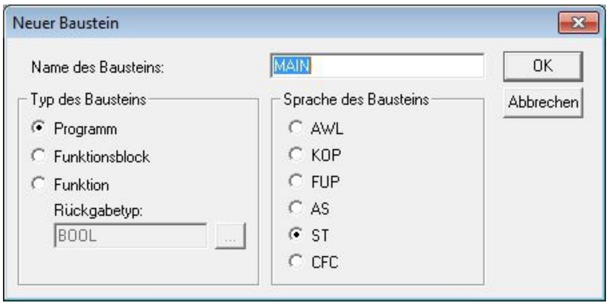
**Create a PLC project as follows:**

1. In the Start menu, right-click on the TwinCAT symbol.
2. In the context menu click on **PLC Control**.

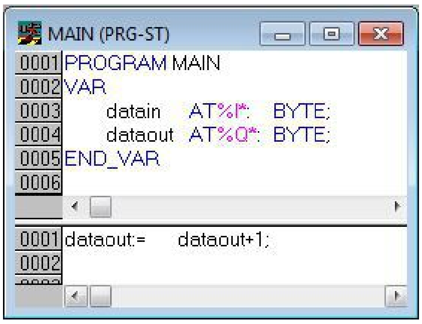



The TwinCAT PLC Control window appears.

3. In the menu click on **File > New** and select the option **PC or CX (x86)**.
4. Under **Block type** select the option **Program**, and under block language select the option **ST (Structured Text)**.




5. Write a small program.




6. Save the PLC project and click on **Project > Compile** in the menu.

⇨ Once the project has been compiled, a file with the extension .tpy is created in the same location as the project file. The file name of the new file is the same as the file name of the PLC project.

In the next step you can add the compiled PLC project in the TwinCAT System Manager.

**Adding a PLC project**

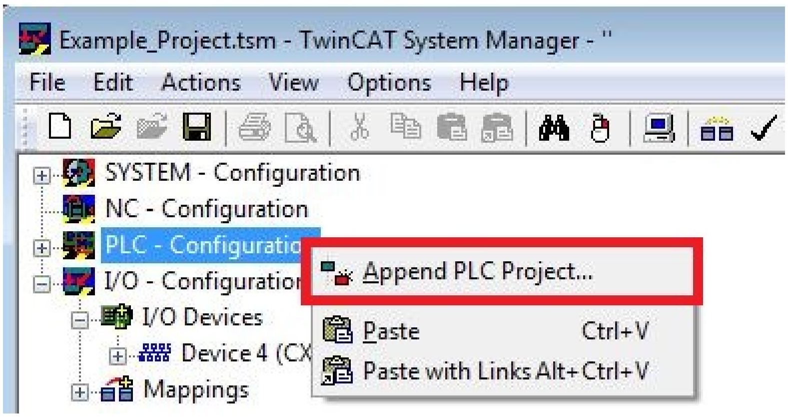
The PLC project can be added in the System Manager. The newly created variables from a PLC project are integrated in the System Manager and can be linked with the inputs and outputs of the hardware.

Prerequisites for this step:

- An Embedded PC, added in TwinCAT.
- A correctly compiled PLC project and a .tpy file.

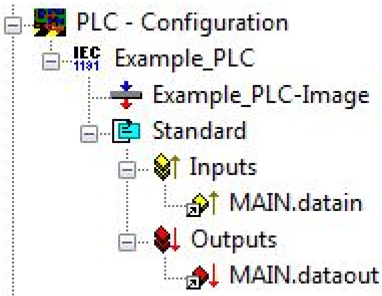
**Proceed as follows:**

1. Switch back to the System Manager window.
2. Right-click on **PLC – Configuration** in the tree view on the left.
3. In the context menu click on **Append PLC Project**.




4. Select a file with the extension .tpy in your system directory and confirm with **OK**.

   The PLC project is added in the tree view under PLC – Configuration. The variables defined in the project are shown under the inputs and outputs.




In the next step you can link the variables with the hardware.

## 6.3.10 Linking variables

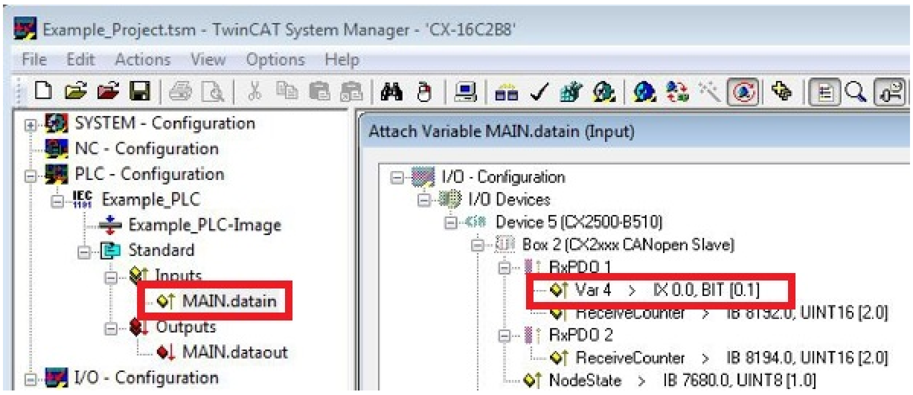
Once the PLC project was successfully added in the System Manager, you can link the newly created input and output variables from the PLC project with the inputs and outputs of your devices.

Prerequisites for this step:

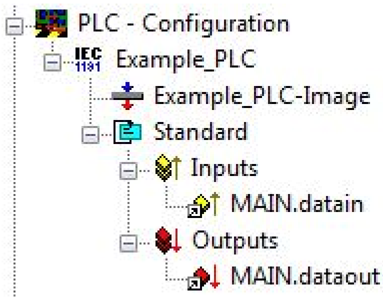
- An added PLC project in the System Manager.

**Link the variables as follows:**

1. Double-click on the input or output variables in the tree view under **PLC - Configuration**.
   The **Attach Variable** window appears and shows which inputs or outputs can be linked with variables.

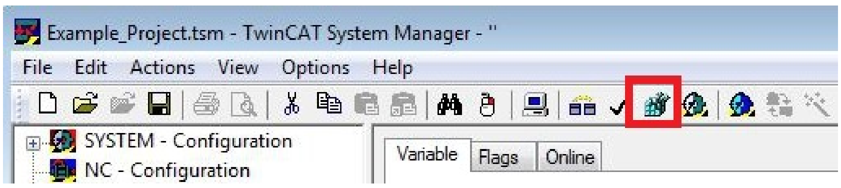



2. Double-click on the inputs or outputs in the Attach Variable window.
   The input variables are linked with the inputs of your hardware, and the output variables with the outputs.




   Variables that are already linked are indicated with a small arrow icon in TwinCAT.

3. In the toolbar click on **Activate Configuration**.




4. Confirm the request whether TwinCAT is to start in Free Run mode with **Yes**.

⇨ You have successfully linked variables with the hardware. Use Activate Configuration to save and activate the current configuration.

The configuration can now be loaded on the CX, in order to automatically start TwinCAT in Run mode, followed by the PLC project.

## 6.3.11 Load configuration to CX

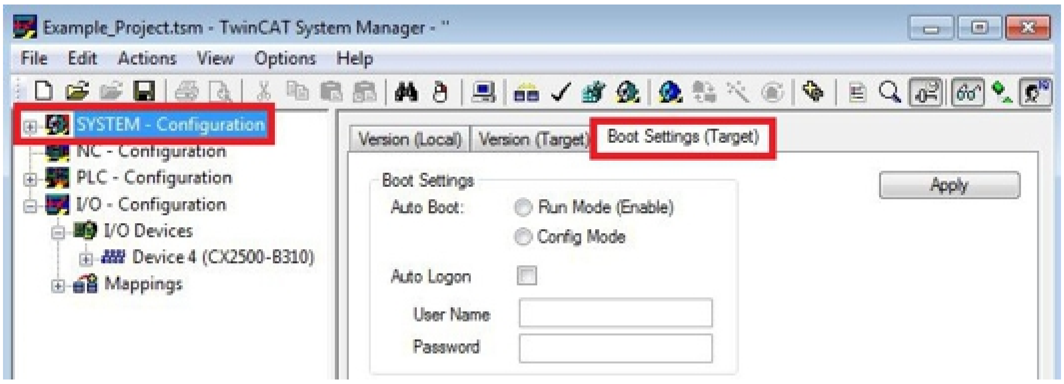
Once all variables are linked, the configuration can be saved and loaded on the CX. This has the advantage that the PLC project is loaded and started automatically when the CX is switched on. The start of the previously created PLC project can thus be automated.

Prerequisites for this step:

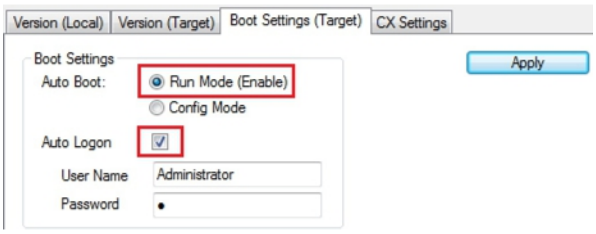
- A completed PLC project, added in the System Manager.
- Variables from the PLC project, linked with the hardware in the System Manager.
- A CX selected as target system.

**Load the configuration on the CX as follows:**

1. In the tree view on the left click on **SYSTEM – Configuration**.
2. Click on the **Boot Settings (Target)** tab.



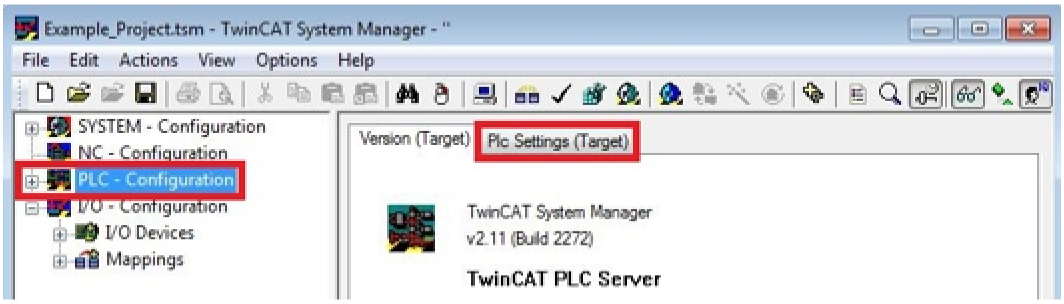

3. Under Boot Settings select the option **Run Mode (Enable)** and tick the **Auto Logon** checkbox.



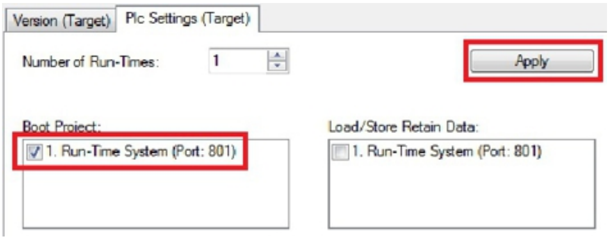

4. Enter the user name and password for the CX in the **User Name** and **Password** fields.
5. Click on **Apply**.
   The Logon Information window appears.


6. Re-enter the user name and the password and click **OK**.

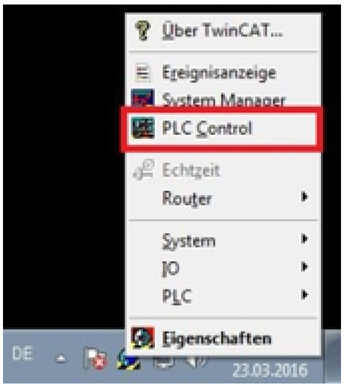
7. In the tree view on the left click on **PLC – Configuration**, then on the **PLC Settings (Target)** tab.




8. Select the Start PLC under Boot Project and click on **Apply**.

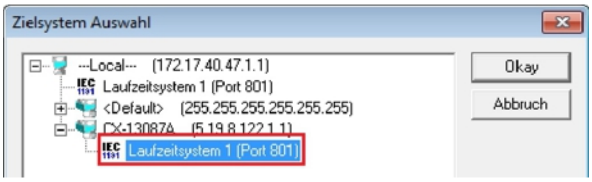



9. Start PLC Control and open the PLC project.




10. In the menu bar at the top click on **Online**, and then on **Choose Runtime System**.

11. Select the runtime system from the CX and click on **OK**.




12. In the menu bar at the top click on **Online**, then **Login**.
    The PLC project is logged in.

13. In the menu bar at the top click on **Online**, then **Create Boot Project**.

⇨ You have successfully loaded the CX configuration. From now on, TwinCAT will start in Run mode and the PLC project will start automatically.

Next, the master can be added in a new project in the System Manager and can then be used to find slaves that have already been set up.

## 6.3.12 Adding a CANopen master

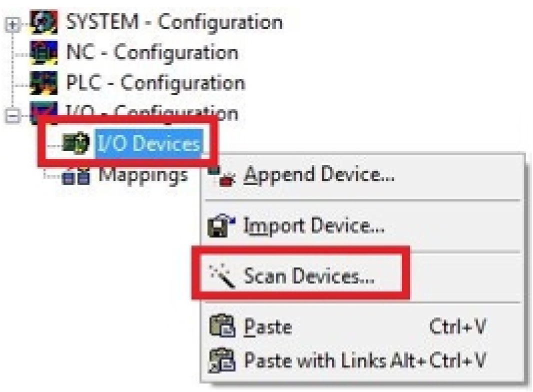
The CANopen master is added with the TwinCAT System Manager, like the other devices. The attached master can then be used to find all connected slaves. The following section illustrates how to add a CANopen master in TwinCAT.

Prerequisites for this step:

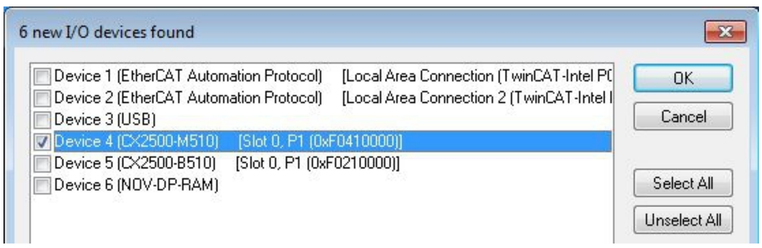
- TwinCAT must be in Config mode.
- A selected target system (in this sample it is the Embedded PC CX2020-M510)

**Add a CANopen device as follows:**

1. Start the System Manager.
2. In the tree view on the left, right-click on **I/O Devices**.
3. In the context menu click on **Scan Devices**.

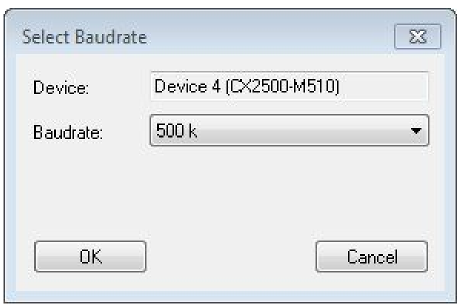



4. Select the devices you want to use and confirm the selection with **OK**.

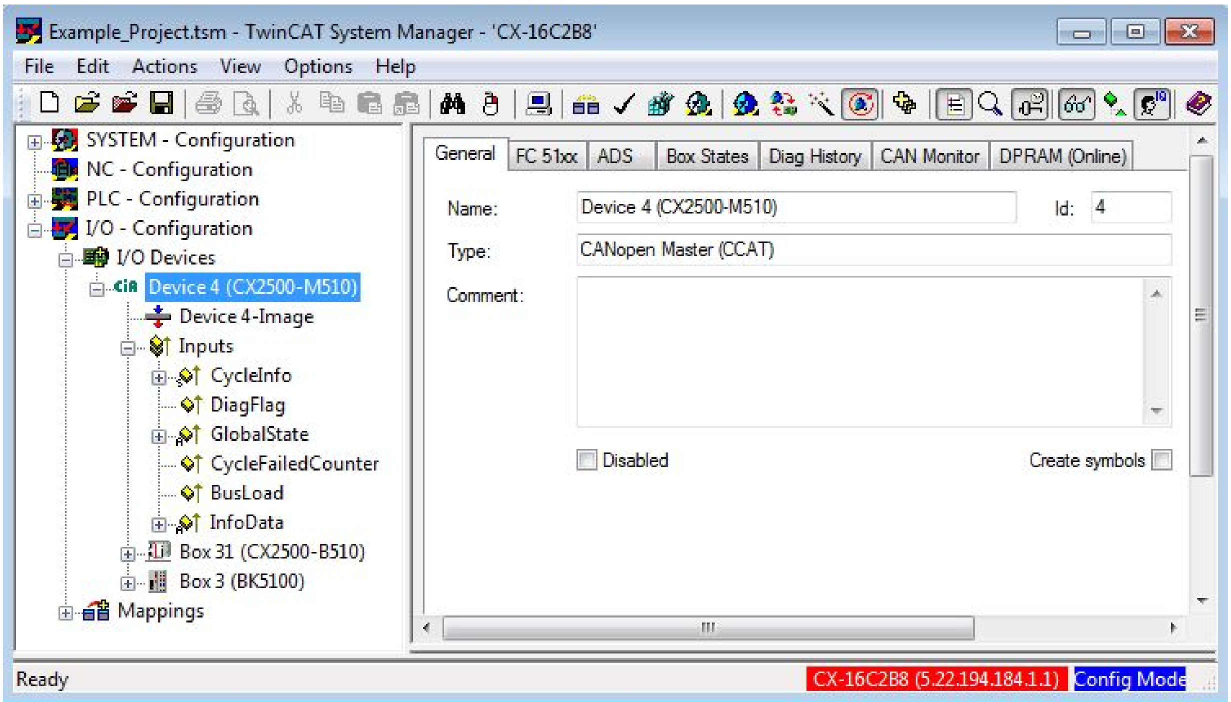



5. Confirm the request with Yes, in order to look for boxes.
   The **Select Baud Rate** window appears.

6. Under Baud Rate select the appropriate baud rate for the CANopen master.



⇨ All devices and slave boxes that are found are displayed in the tree view on the left, including Bus Terminals connected to the devices or slave boxes.




Repeat the steps if not all devices are displayed. If not all devices and slave boxes are found despite the repeat operation, check the cabling of the devices and slave boxes.

## 6.4 Parameterization with TwinCAT 3

This section illustrates how CANopen devices can be parameterized with the aid of TwinCAT 3. A total of three devices are used for the sample, including a CANopen master, to which two CANopen slaves are connected.

First, the process of finding and selecting a target system in TwinCAT is illustrated. Next, a CANopen slave is added and parameterized in TwinCAT, and the CANopen address of the slave is set. Then a PLC project is created and added in TwinCAT. Then, the variables from the PLC project are linked with the hardware, and the finished configuration is loaded on the CANopen slave.

In the last step, the CANopen master is added in TwinCAT, and the two CANopen slaves are located via the master.

### 6.4.1 Searching for target systems

Before you can work with the devices, you must connect your local computer to the target device. Then you can search for devices with the help of the IP address or the host name.

The local PC and the target devices must be connected to the same network or directly to each other via an Ethernet cable. In TwinCAT a search can be performed for all devices in this way and project planning subsequently carried out.

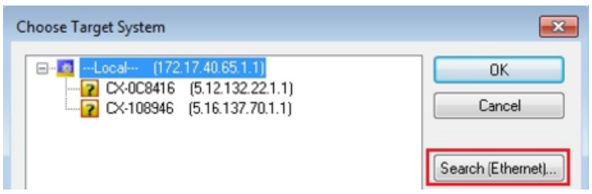
Prerequisites for this step:

- TwinCAT 3 must be in Config mode.
- IP address or host name of the device.

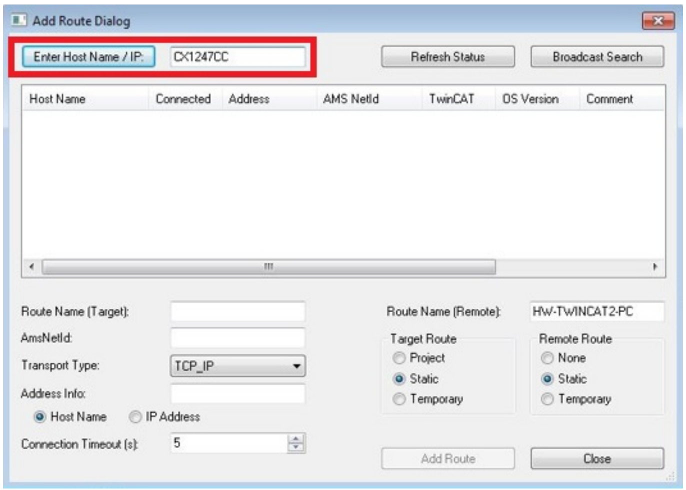
**Search for the devices as follows:**

1. In the menu at the top click on **File > New > Project** and create a new TwinCAT XAE project.
2. In the tree view on the left click on **SYSTEM**, and then **Choose Target**.

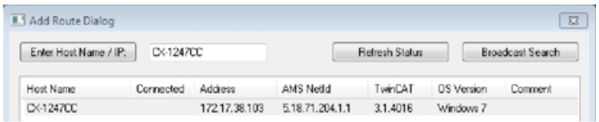

3. Click on **Search (Ethernet)**.



4. Type the host name or the IP address of the device into the **Enter Host Name / IP** box and press **[Enter]**.




5. Mark the device found and click on **Add Route**.

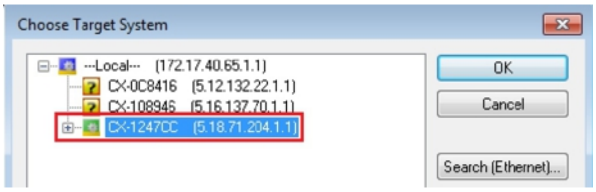



The Logon Information window appears.
Enter the user name and password for the CX in the **User Name** and **Password** fields and click **OK**.

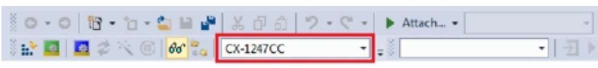

The following information is set as standard in CX devices:
**User name:** Administrator **Password:** 1

6. If you do not wish to search for any further devices, click on **Close** to close the Add Route Dialog. The new device is displayed in the Choose Target System window.

7. Select the device you want to specify as target system and click **OK**.




⇨ You have successfully searched for a device in TwinCAT and inserted the device as the target system. The new target system and the host name are displayed in the menu bar.




Using this procedure you can search for all available devices and also switch between the target systems at any time. Next, you can append the device to the tree view in TwinCAT.

## 6.4.2 Add CANopen slave

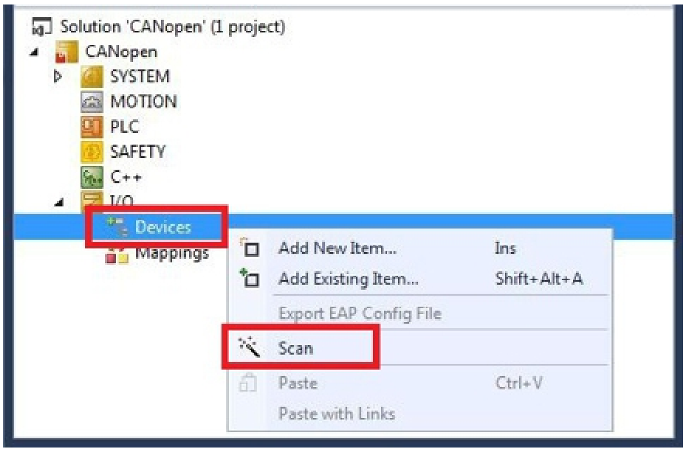
The sample shows a CX2020 CANopen slave with CX2500-B510 fieldbus module, connected to the CANopen master. In order to ensure that the CANopen slave is configured and subsequently detected by the CANopen master with all inputs and outputs, the CANopen slave first must be added in TwinCAT.

Prerequisites for this step:

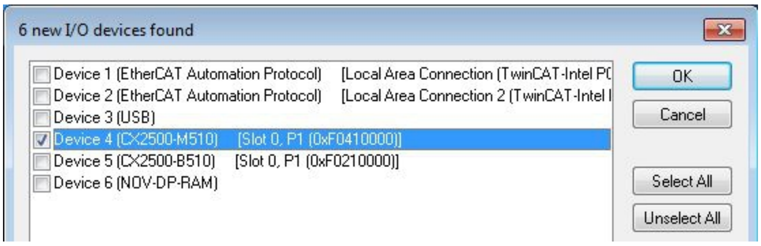
- A scanned and selected target device with CANopen slave. This sample uses a CX2020 with CX2500-B510 fieldbus module.

**Add the CANopen slave as follows:**

1. In the tree view on the left, right-click on **Devices**.
2. In the context menu click on **Scan**.

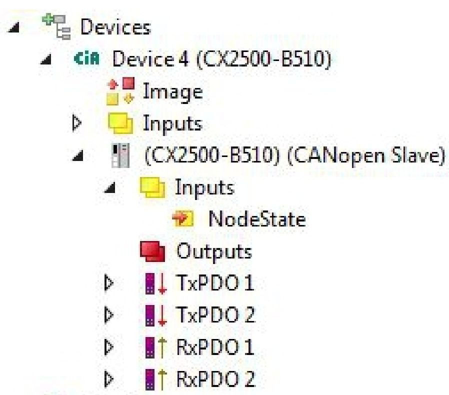



3. Select the devices you want to use and confirm the selection with **OK**.




4. Confirm the request with Yes, in order to look for boxes.

⇨ The CANopen slave was successfully added in TwinCAT 2 and is displayed in the tree view with the inputs and outputs.




In the next step you can extend the process image by creating additional virtual slaves. Or you can set the address, once the slave configuration is complete.

## 6.4.3 Creating a virtual slave

Additional virtual slaves can be created on the same hardware interface. This enables more data to be exchanged with a CANopen master, or a connection with a second CANopen master can be established.

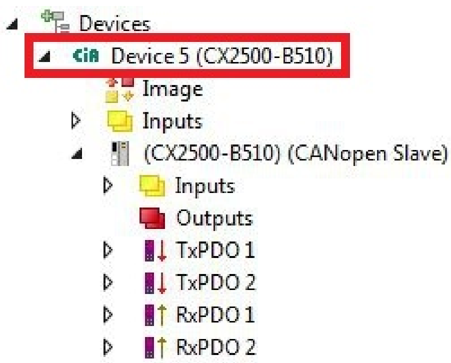
Each virtual slave is assigned a dedicated address via TwinCAT and is configured like an independent device for the CANopen master.

Prerequisites for this step:

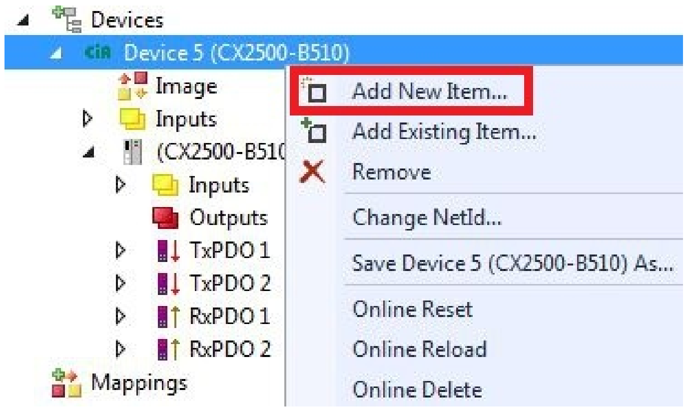
- A CANopen slave, created in TwinCAT.

**Create a virtual slave as follows:**

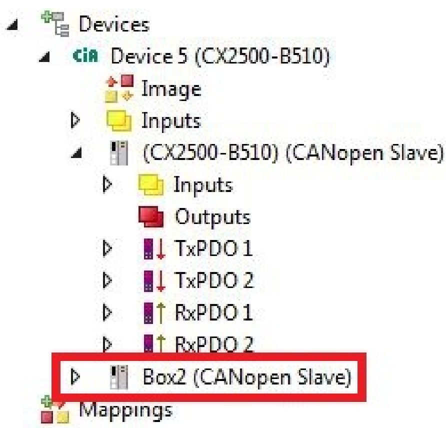
1. Right-click on a CANopen slave in the tree view on the left.




2. Click on **Add New Item** in the context menu.




⇨ A further box (virtual slave) is created.




Variables for the virtual slave can now be created. In the next step you can set the address for the slave.

## 6.4.4 Setting the address

Once the CANopen slave was successfully added in TwinCAT, the address of the CANopen slave can be set. Devices with a DIP switch have a preset address. The address on the DIP switch must match the address set in TwinCAT.

For devices without DIP switch the address is only set in TwinCAT.

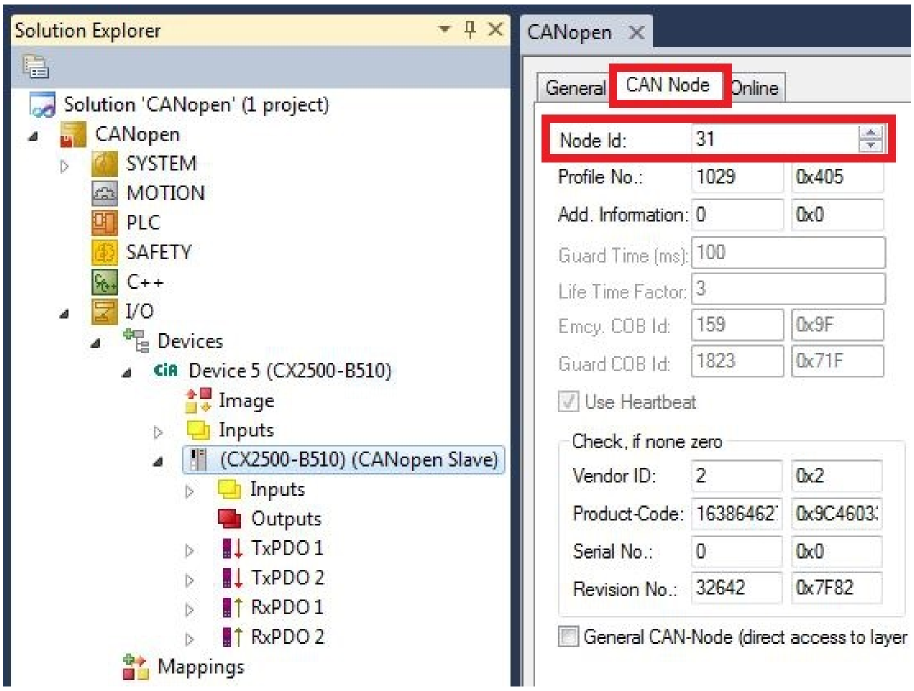
In this step the address is set in TwinCAT, so that the CANopen slave can be reached by the CANopen master via this address.

Prerequisites for this step:

- An added CANopen slave in TwinCAT.

**Parameterize the CANopen slave as follows:**

1. Click on a slave box.
2. Click on the **CAN Node** tab.
3. Enter a value for the CANopen address in the **Node Id** field, e.g. “31”.




⇨ You have set the address successfully. The CANopen master can reach the CANopen slave with the set address.
You can now create further PDOs.

## 6.4.5 Setting the address with a DIP switch

To ensure that the address set via the address selection switch is used, the corresponding setting must be enabled in TwinCAT.

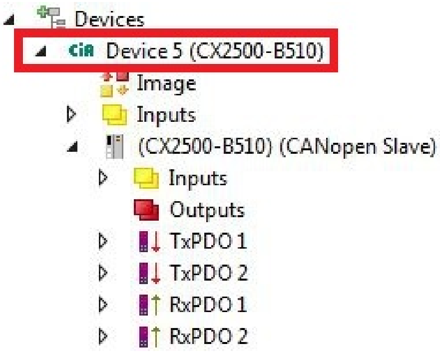
The corresponding setting in TwinCAT is only active, if the CANopen device has an address selection switch.

Prerequisites for this step:

- CANopen device with address selection switch.
- CANopen device added in TwinCAT.

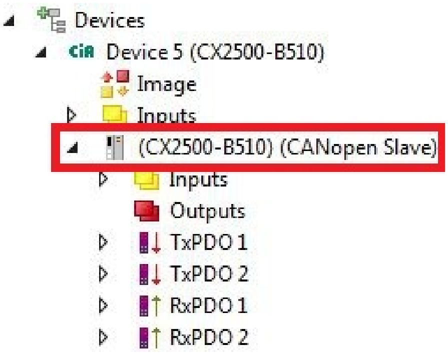
**Set the slave node id as follows:**

1. Click on the CANopen device in the tree view on the left.

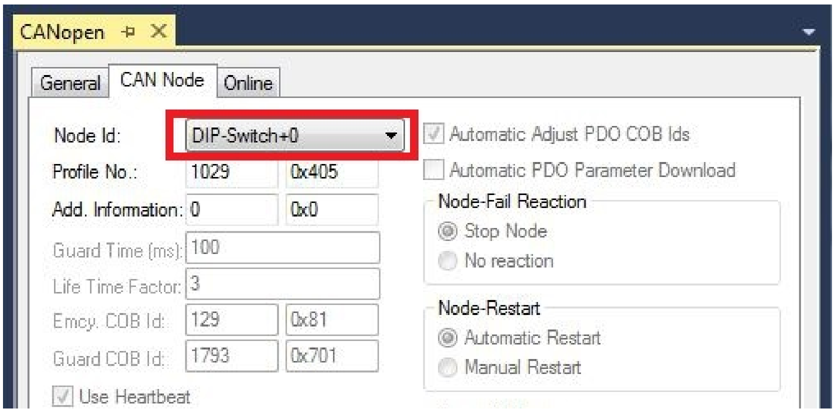



2. Click on the **FC 51xx** tab and select the option **Set by DIP-switch** under **Slave Node-ID**.


3. Click on the CANopen box in the tree view on the left.



4. Click on the **CAN Node** tab and select the option **DIP-Switch+0** under **Node Id**.
   In this example the number 0 is added to the address set on the DIP switch. Other numbers (0-126), e.g. for virtual slaves, may also be added to the address.




⇨ You have set the slave node id successfully, and the CANopen slave will use the set address from now on. You can now create further PDOs.

## 6.4.6 Creating further PDOs

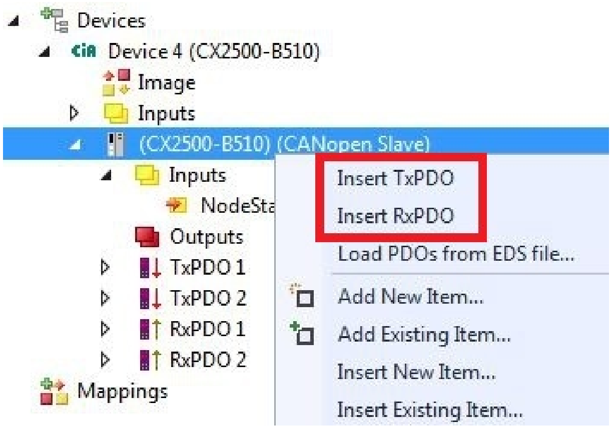
The CANopen slave can exchange up to 16 PDOs (each with 8 bytes of process data) with the CANopen master in input and output direction. By default 2 PDOs are created in Tx and Rx direction.
Here we shown how to create further PDOs for a CANopen slave.

Prerequisites for this step:

- A CANopen slave added in the tree view.

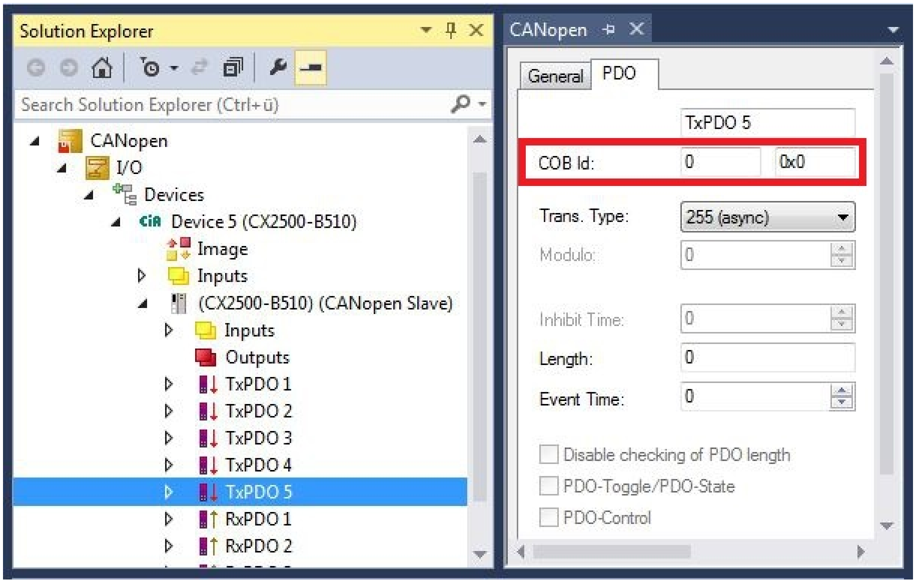
**Create the PDOs as follows:**

1. Right-click on a CANopen slave in the tree view.
2. In the context menu, click on **Insert TxPDO** or **Insert RxPDO** in order to create PDOs in Tx or Rx direction.




The new TxPDOs or RxPDOs are added in the tree view below the already created PDOs and numbered consecutively.

**Notice** From the fifth PDO in Tx or Rx direction the COB ID is no longer entered automatically (see the following diagram).




3. From the fifth PDO in Tx or Rx direction click on the **PDO** tab.
4. Enter the required value in the **COB ID** field.

⇨ You have successfully created further PDOs; in the next step you can create variables for the data exchange under the PDOs.

## 6.4.7 Creating variables

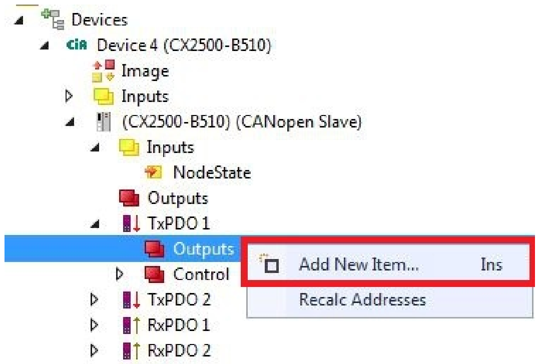
In TwinCAT the PDOs are filled with variables, which can later be linked with the PLC program. This section describes how to create variables.

Prerequisites for this step:

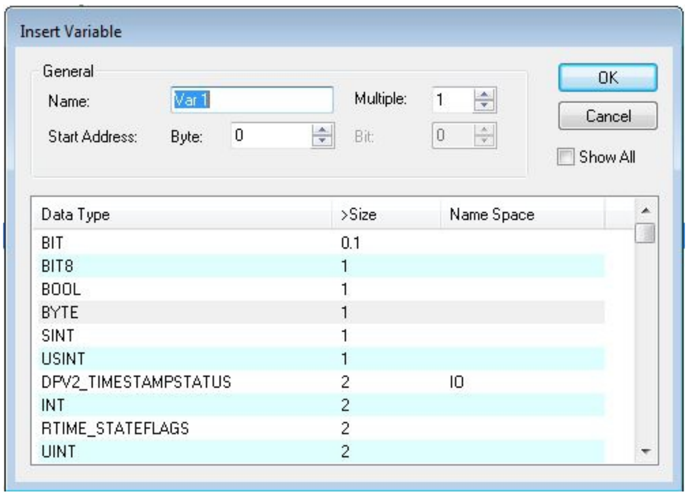
- Newly created PDOs, which are to be filled with variables.

**Create the variables as follows:**

1. In the tree view click on a TxPDO or RxPDO to show more information.
2. Right-click on Outputs or Inputs, depending on whether a TxPDO or RxPDO is selected.




3. Click on **Add New Item** in the context menu.
   The **Insert Variable** window appears.
4. Click on the required variable, then click **OK**.




⇨ You have successfully created a variable. The new variable is shown in the tree view on the left. In this way you can add further variables for the CANopen slave.
In the next step you can specify the transmission type, thereby specifying how the process data objects are transferred.

## 6.4.8 Setting the transmission type

The transmission type determines how the process data objects are transferred. The transmission type for the RxPDOs and TxPDOs is set on the PDO tab.

The available transmission types are: acyclic synchronous, cyclic synchronous, and asynchronous.

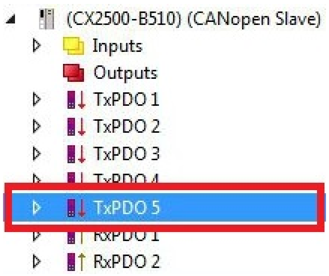
| | | | |
|---|---|---|---|
| Transmission type: | Acyclic Synchronous | Cyclic Synchronous | Asynchronous |
| Name in TwinCAT: | (acyc, sync) | (cyc, sync) | (async) |

Prerequisites for this step:

- A CANopen slave with process data objects (PDO) added in TwinCAT

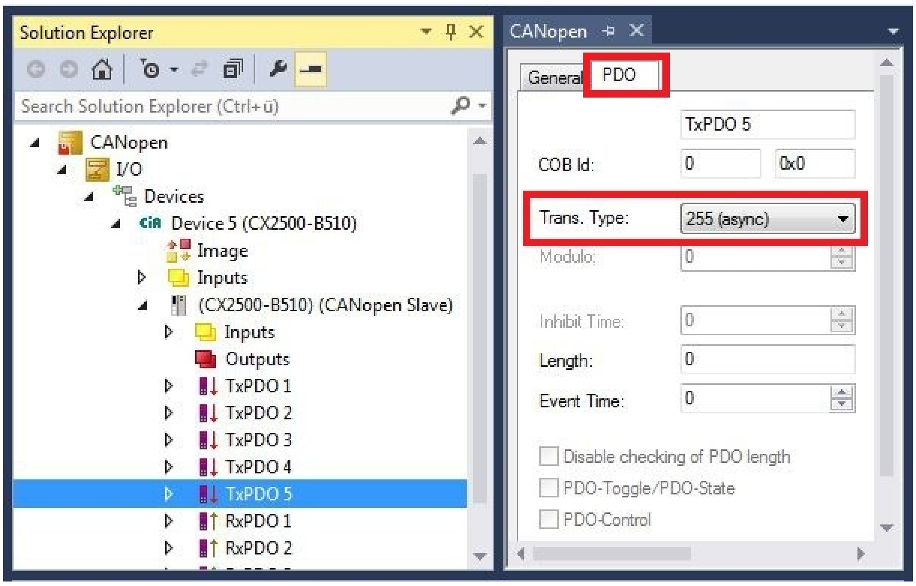
**Specify the transmission type as follows:**

1. In the tree view, left-click on a process data object (PDO).




2. Click on the **PDO** tab.

3. Select the required transmission type under **Trans. Type**.




⇨ You have successfully specified a transmission type for a process data object. The transmission types for the remaining process data objects are specified in the same way.
Next, you can create a PLC project for the CANopen slave.

## 6.4.9 Creating a PLC project

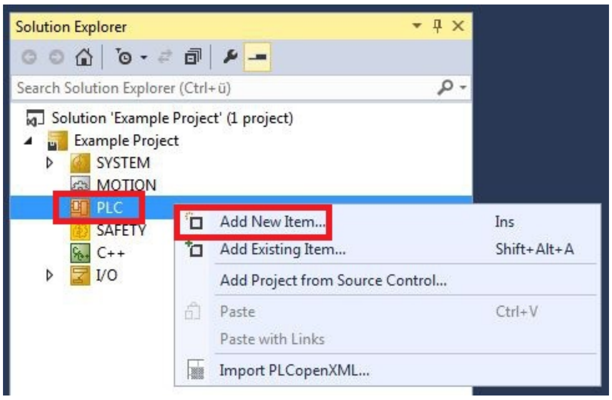
The next steps describe how to create a PLC project in TwinCAT and add it in the tree view.

Prerequisites for this step:

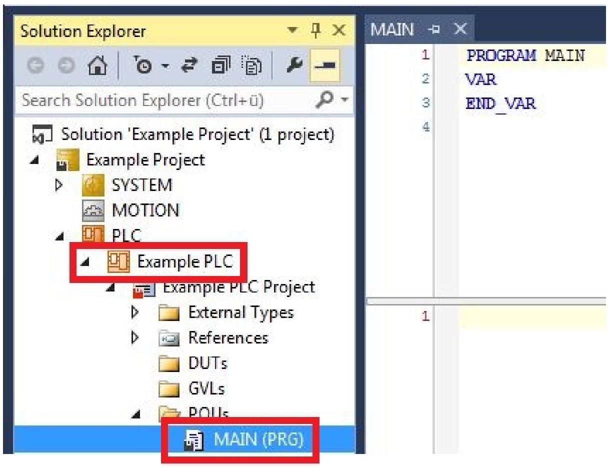
- A newly created TwinCAT XAE project.

**Create a PLC project as follows:**

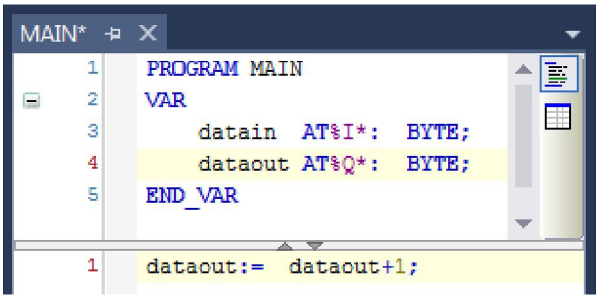
1. Right-click on **PLC** in the tree view.
2. In the context menu click on **Add New Item** and select the **Standard PLC Project**.



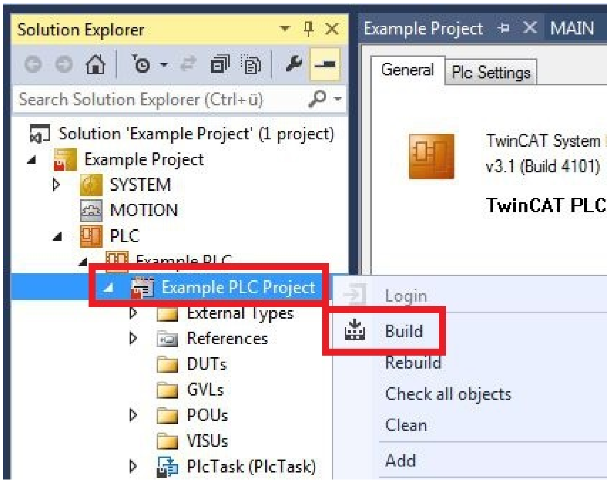

3. In the tree view click on the newly created PLC project, then double-click on **MAIN (PRG)** under **POUs**.



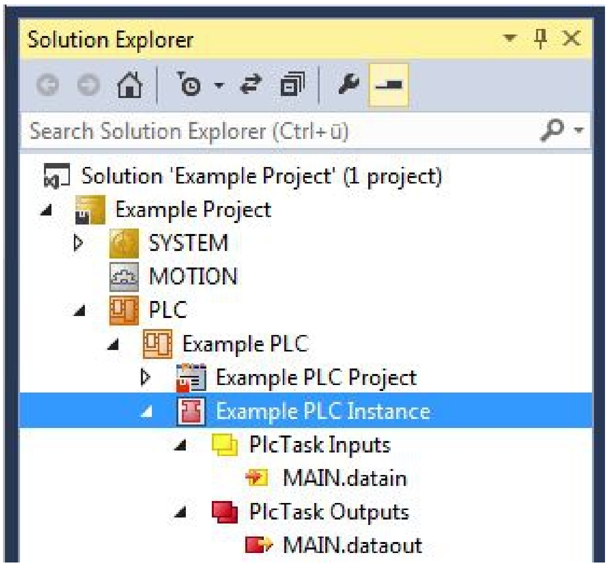

4. Write a small program, as shown in the diagram below.



5. In the tree view right-click on the PLC project, then click on **Build** in the context menu.




⇨ You have successfully created a PLC project and added the project in TwinCAT. A PLC instance with the variables for the inputs and outputs is created from the PLC project.




In the next step you can link the variables with the hardware.

## 6.4.10 Linking variables

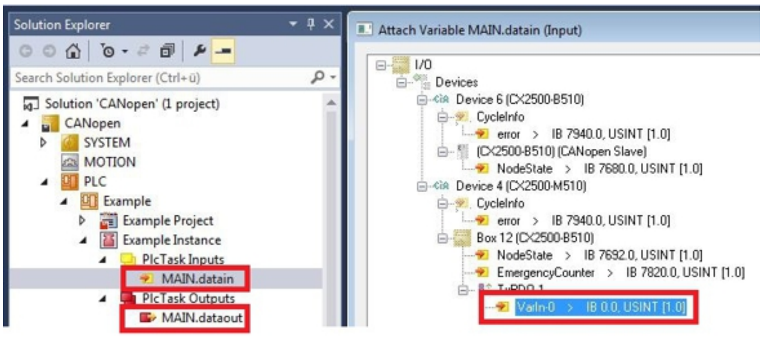
Once the PLC project was successfully added in the System Manager, you can link the newly created input and output variables from the PLC project with the inputs and outputs of your hardware.

Prerequisites for this step:

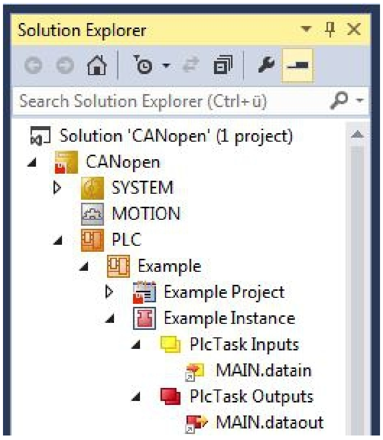
- A PLC program attached in TwinCAT.

**Link the variables as follows:**

1. Double-click on the input or output variables in the tree view under **PLC**.
   The **Attach Variable** window appears and shows which inputs or outputs can be linked with the variables from the PLC project.

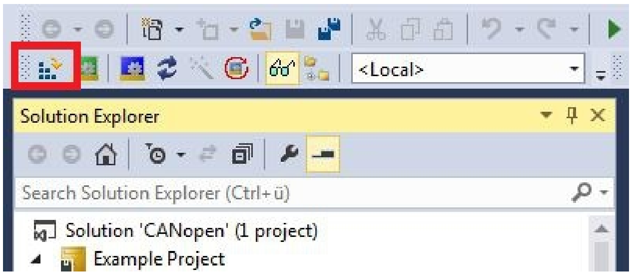


2. Double-click on the inputs or outputs of the hardware in the **Attach Variable** window.
   Link the input variables with the inputs and the output variables with the outputs of the hardware.



   Variables that are already linked are indicated with a small arrow icon in TwinCAT.

3. In the toolbar click on **Activate Configuration**.



4. Confirm the request whether TwinCAT is to start in Free Run mode with **Yes**.

⇨ You have successfully linked variables with the hardware. Use Activate Configuration to save and activate the current configuration.

The configuration can now be loaded on the CX, in order to automatically start TwinCAT in Run mode, followed by the PLC project.

## 6.4.11 Load configuration to CX

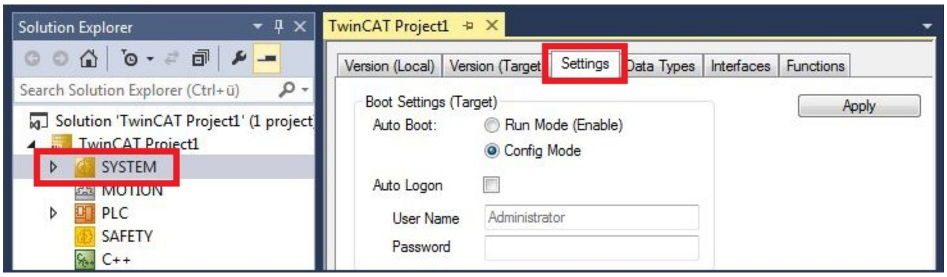
Once variables are linked, the configuration can be saved and loaded on the CX. This has the advantage that the PLC project is loaded and started automatically when the CX is switched on. The start of the previously created PLC project can thus be automated.

Prerequisites for this step:

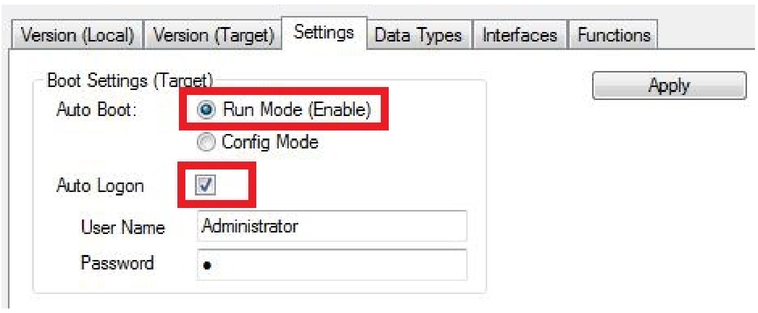
- A completed PLC project, added in the System Manager.
- Variables from the PLC project, linked with the hardware in the System Manager.
- A CX selected as target system.

**Load the configuration from the System Manager to the CX as follows:**

1. In the tree view on the left click on **SYSTEM**.
2. Click on the **Settings** tab.

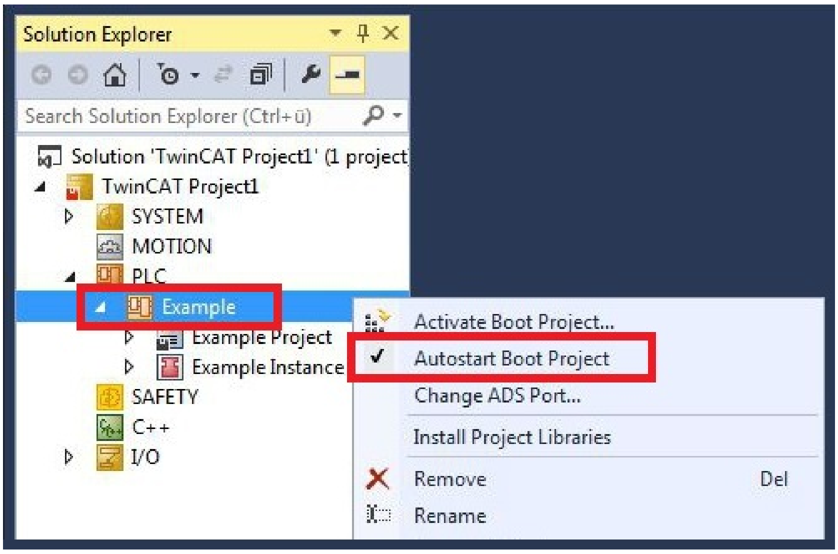



3. Under Boot Settings select the option **Run Mode (Enable)** and tick the **Auto Logon** checkbox.



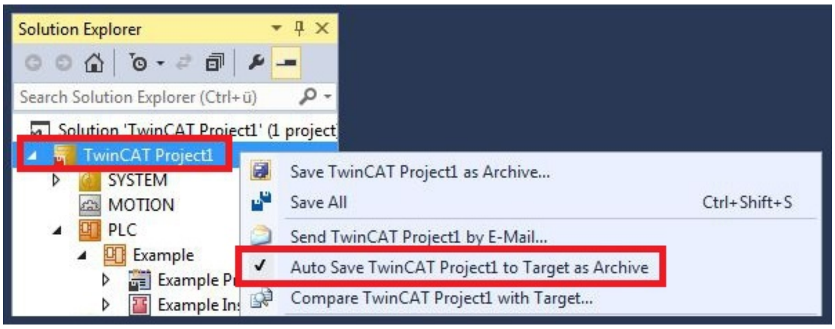

4. Enter the user name and password for the CX in the **User Name** and **Password** fields.
5. Click on **Apply**.
6. In the tree view on the left right-click on the PLC project under **PLC**.

7. In the context menu click on **Autostart Boot Project**.
   The setting is selected




8. Right-click on the project folder in the tree view.

9. In the context menu click on **Auto Save to Target as Archive**.
   The setting is selected.




⇨ You have successfully loaded the CX configuration. From now on, TwinCAT will start in Run mode and the PLC project will start automatically.

Next, the master can be added in a new project in the System Manager and can then be used to find slaves that have already been set up.

## 6.4.12 Adding a CANopen master

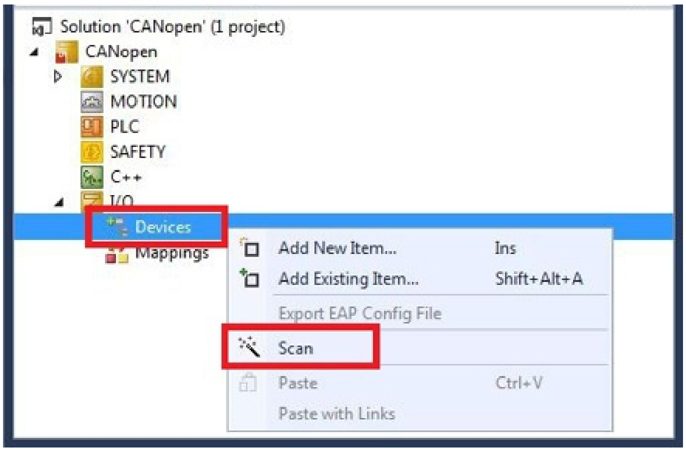
The CANopen master is added with the TwinCAT System Manager, like the other devices. The attached master can then be used to find all connected slaves. The following section illustrates how to add a CANopen master in TwinCAT.

Prerequisites for this step:

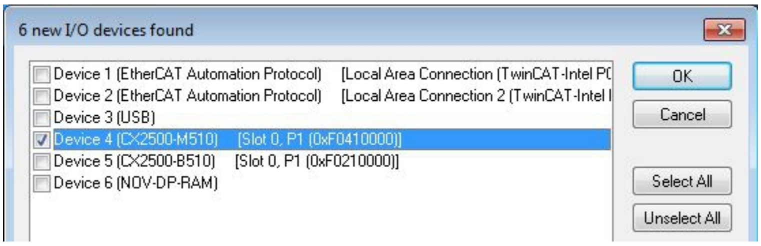
- TwinCAT must be in Config mode.
- A selected target system (in this sample it is the Embedded PC CX2020-M510)

**Add a CANopen device as follows:**

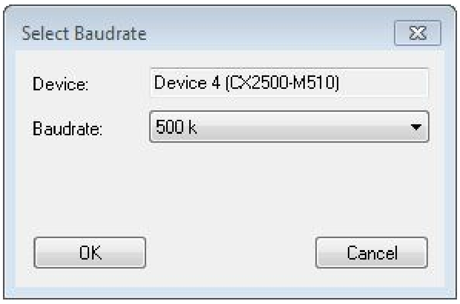
1. In the tree view on the left, right-click on **Devices**.
2. In the context menu click on **Scan**.



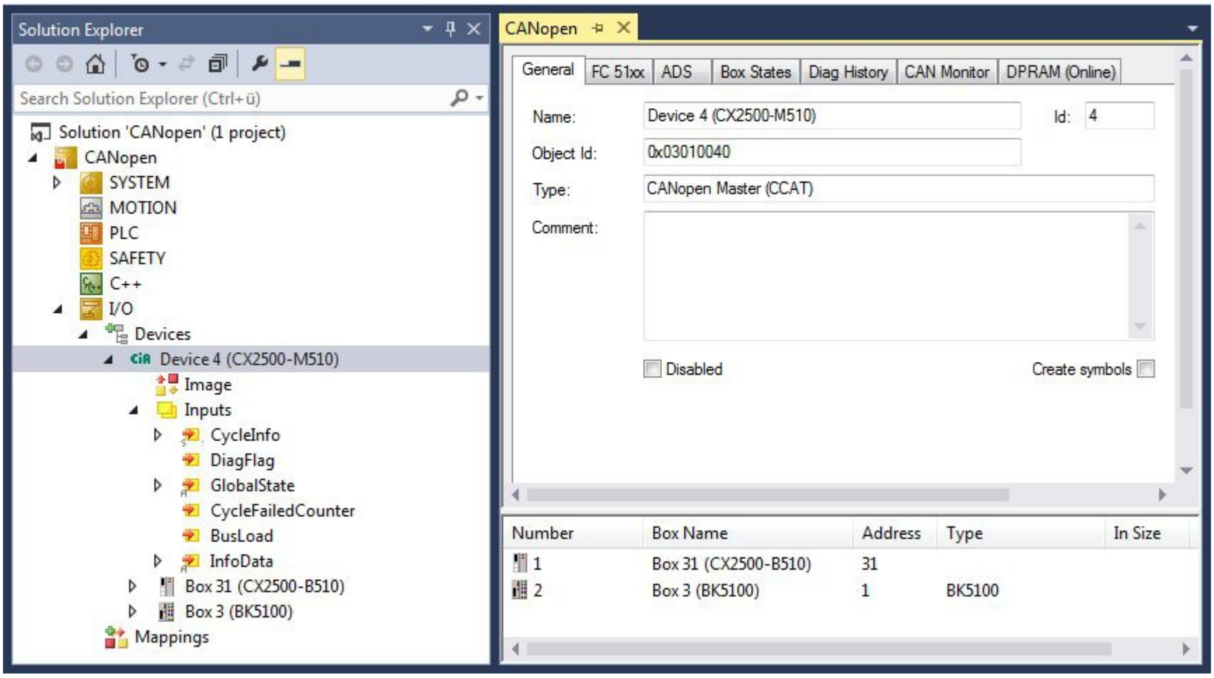

3. Select the devices you want to use and confirm the selection with **OK**.




4. Confirm the request with Yes, in order to look for boxes.
   The **Select Baud Rate** window appears.
5. Under Baud Rate select the appropriate baud rate for the CANopen master.



⇨ All devices and slave boxes that are found are displayed in the tree view on the left, including Bus Terminals connected to the devices or slave boxes.




Repeat the steps if not all devices are displayed. If not all devices and slave boxes are found despite the repeat operation, check the cabling of the devices and slave boxes.

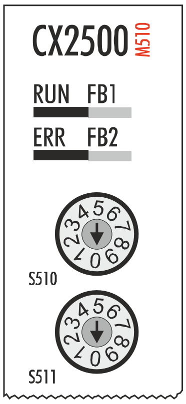
# 7 Error handling and diagnostics

## 7.1 Diagnostic LEDs

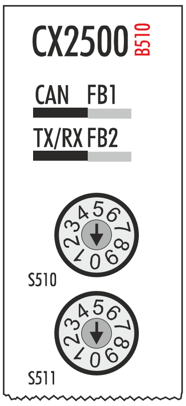
The diagnostic LEDs of a CANopen master and CANopen slaves are described here. The labelling of the diagnostic LEDs on a CX2500 fieldbus module and an Embedded PC with optional interface is identical.

The LED description therefore only distinguishes between CANopen master and CANopen slave.

**M510 (master)**

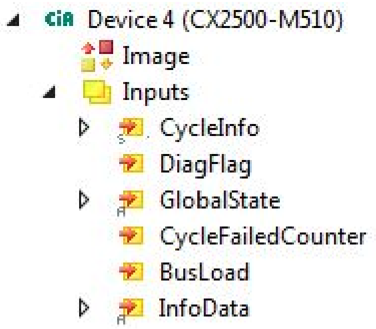
| Display | LED | Color | Meaning |
|---|---|---|---|
|  | RUN | Green | CAN is OK |
| | | Red | CAN in BUS off |
| | | Green 200 ms / Red 200 ms | CAN warning |
| | | Green and red flashing rapidly | Baud rate search active |
| | | Green off / Red on | CAN not configured |
| | ERR | Green on / Red off | All nodes have NodeState = 0 |
| | | Green 200 ms / Red 200 ms | All boxes in OP state, but the tasks have not yet started |
| | | Green off / Red 200 ms | Not all nodes in OP |
| | | Green off / Red on | No boxes configured |

**B510 (slave)**

| Display | LED | Color | Meaning |
|---|---|---|---|
|  | CAN<br>Shows the CAN status | Green | CAN is OK |
| | | Red | CAN in BUS off |
| | | Green 200 ms / Red 200 ms | CAN warning |
| | | Green and red flashing rapidly | Baud rate search active |
| | | Green off / Red on | CAN not configured |
| | Tx/Rx<br>Indicates CAN errors | Green on | Everything OK |
| | | Green 200 ms / Red 200 ms | All boxes in OP state, but the task has not yet started |
| | | Red 200 ms | Not all boxes in OP |
| | | Green off / Red on | No boxes configured |

## 7.2 Status messages

The CANopen status messages provide additional information and can be used for diagnostic purposes.



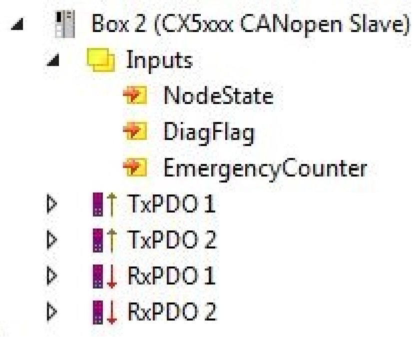

The following table shows which values the variables can assume:

| Inputs | Meaning |
| --- | --- |
| CycleInfo | Cycle Counter:<br>This counter is incremented by one after each cycle.<br>Error:<br>Shows the number of boxes, with a non-zero BoxState.<br>ActualCycle Time:<br>Reserved for future use |
| DiagFlag | This variable provides information on changes to the diagnostic data.<br>• 0: Data unchanged.<br>• 1: Data changed. Use ADS Read to read the data. |
| GlobalState | This variable provides information on the status of the master.<br>GlobalState[0]:<br>0: Device is in RUN status.<br>1: Device is in RESET status.<br>2: Device is in OFFLINE status.<br>3: Device is in STOP status.<br>GlobalState[1] (FW V02.14 or higher):<br>Bit 0-7: RxError counter of the CAN controller<br>Bit 8-15: TxError counter of the CAN controller<br>GlobalState[2]:<br>Bit 0: CAN controller is in BUS OFF.<br>Bit 1: CAN controller warning limit reached.<br>Bit 2: Rx queue exceeded.<br>Bit 3: Hi-Prio Tx queue exceeded.<br>Bit 4: Lo-Prio Tx queue exceeded.<br>Bit 5: CAN send error (FW V02.14 or higher).<br>Bit 6-14: Reserved for future use.<br>Bit 15: Switches with each sent SYNC message.<br>GlobalState[3]:<br>Bus load in %. |
| CycleFailedCounter | This counter is incremented by one whenever the last bus cycle is incomplete at the start of a TwinCAT cycle. |
| BusLoad | Bus load in %. |
| InfoData | |

## 7.3 Communication

In the tree view, input variables are listed under the **Inputs** menu item, which provide information about a CANopen device.

The NodeState variable can be used to show the state of the CANopen communication, to indicate whether the slave is in data exchange or an error is present.




The following table shows which values the variable **NodeState** can assume:

| Value | Meaning |
|---|---|
| 0 | No error |
| 1 | Node deactivated |
| 2 | Node not found |
| 4 | SDO syntax error at StartUp |
| 5 | SDO data mismatch at StartUp |
| 8 | Node StartUp in progress |
| 11 | FC510x Bus-OFF |
| 12 | Pre-Operational |
| 13 | Severe bus fault |
| 14 | Guarding: toggle error |
| 20 | TxPDO too short |
| 22 | Expected TxPDO is missing |
| 23 | Node is Operational but not all TxPDOs were received |
| 31 | only for EtherCAT gateways: WC-State of cyclic EtherCAT frame is 1 |
| 128 | Node is Operational but not all RxPDOs were received |
| 129 | Node is Pre-Operational |
| 130 | Node is Stopped |

The following table shows which values the variable **DiagFlag** can assume. This variable provides information on changes to the diagnostic data.

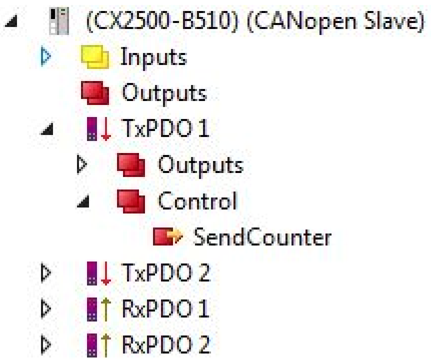
| Value | Meaning |
|---|---|
| 0 | Data unchanged. |
| 1 | Data changed. Use ADS Read to read the data. |

The **EmergencyCounter** variable is incremented by one if an emergency telegram was received.

## 7.4 PDOs

### SendCounter

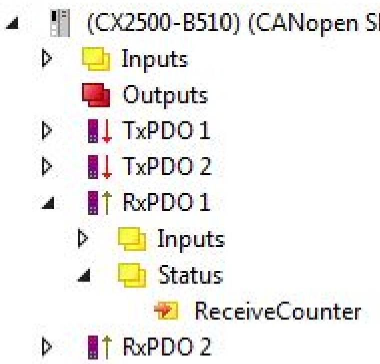
TxPDOs feature an additional SendCounter variable under the **Control** menu item.




The output variable is incremented by one whenever a PDO is sent.

### ReceiveCounter

RxPDOs feature an additional ReceiveCounter variable under the **Status** menu item.




The input variable is incremented by one whenever a PDO is received. In this way newly arrived PDO are always logged, even if the data in the PDO are unchanged. The variable nevertheless indicates whether a device still sends data on a regular basis. It is useful to link variable with the PLC and monitor it.

# 8 Disassembly and disposal

**The disassembly of a CX20x0 hardware configuration with system interfaces takes place in 3 steps**

### 1. Switching off and disconnecting the power supply

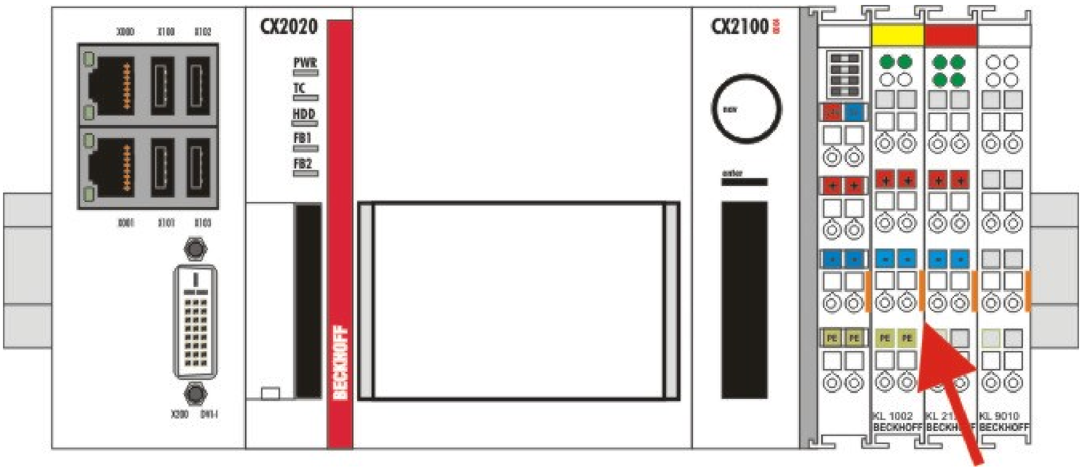
Before a CX20x0 system can be dismantled, the system should be switched off, and the power supply should be disconnected.

### 2. Removing from the DIN rail

Before the individual modules are disconnected, the whole CX20x0 hardware block should be removed from the DIN rail. Proceed as follows:

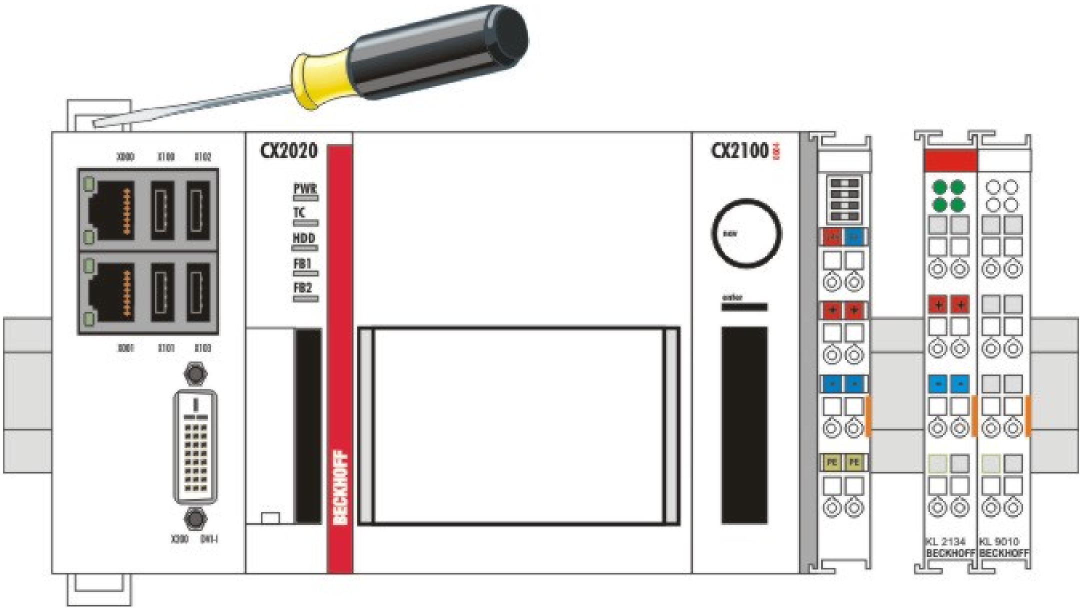
### 2.1. Release and remove the first Terminal next to the power supply unit on the DIN rail.

First remove any wiring from power supply unit *and* then from the first terminal on the DIN rail next to the power supply unit. If the wiring is to be reused for another system, it is advisable to make a note of the connections. Then pull the orange terminal release (see arrow) to release the terminal and pull it out.

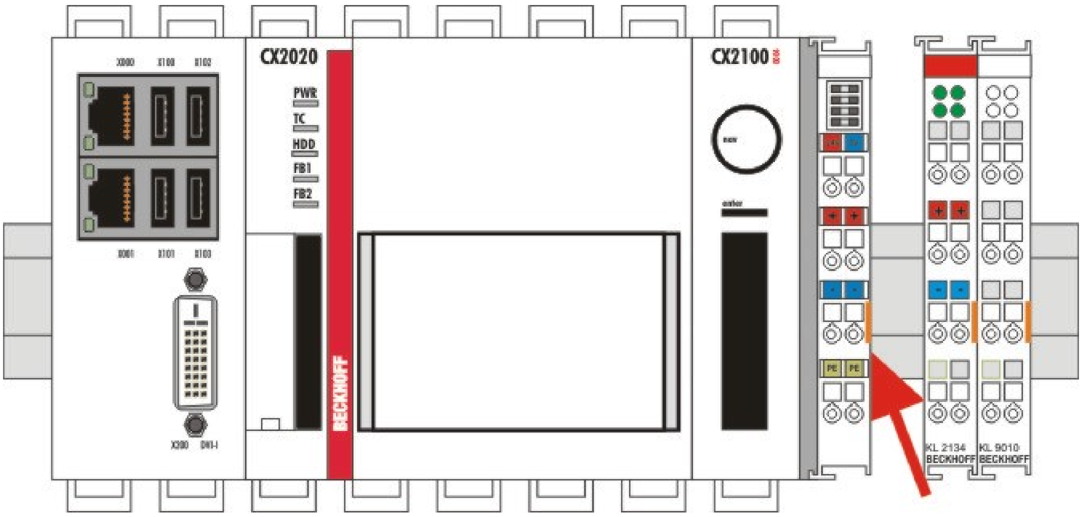



### 2.2. Releasing the CX20x0 systems

In order to release the CX20x0 block, the DIN rail fastening above and below the device must be released. To do this, press the hooks outwards using a screwdriver. An audible click indicates that the device is released.




After pulling on the terminal release of the power supply unit (see arrow) the block can be **carefully** removed from the DIN rail.




**Disposal**

The device must be fully dismantled in order to dispose of it.

Electronic parts must be disposed of in accordance with national electronics scrap regulations.

**3. Disconnecting the system interface**

Disconnecting the system modules from the basic module

If the modules are locked, i.e. attached with tie clips, the clips must be released. To this end lift the tie clips with a screwdriver and pull them out. Subsequently, the system interfaces can be separated again.

**NOTICE**

**Do not use force to open the device!**

Opening the module housing by force would destroy it. The devices may only be opened by Beckhoff service personnel.

# 9 Appendix

## 9.1 Accessories

Cables and connectors for the connection of the CAN components.

**Cable**

| Item number | Description |
|---|---|
| ZB5100 | CAN cable, 4-core, fixed installation 2 x 2 x 0.25 mm², price per meter |

**Connector**

| Item number | Description |
|---|---|
| ZS1051-3000 | Bus interface connector, D-sub for CANopen, in the housing, with switchable termination resistor |
| ZS1052-3000 | Bus interface connector, 5-pin for BK5xxx, in the housing, with switchable termination resistor |
| ZS1052-5150 | CAN diagnostics interface |

## 9.2 Certifications

All products of the Embedded PC family are CE, UL and EAC certified. Since the product family is continuously developed further, we are unable to provide a full listing here. The current list of certified products can be found at www.beckhoff.com.

**FCC Approvals for the United States of America**

**FCC: Federal Communications Commission Radio Frequency Interference Statement**

This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference in which case the user will be required to correct the interference at his own expense.

**FCC Approval for Canada**

**FCC: Canadian Notice**

This equipment does not exceed the Class A limits for radiated emissions as described in the Radio Interference Regulations of the Canadian Department of Communications.

## 9.3 Support and Service

Beckhoff and their partners around the world offer comprehensive support and service, making available fast and competent assistance with all questions related to Beckhoff products and system solutions.

**Beckhoff's branch offices and representatives**

Please contact your Beckhoff branch office or representative for local support and service on Beckhoff products!

The addresses of Beckhoff's branch offices and representatives round the world can be found on her internet pages:
http://www.beckhoff.com

You will also find further documentation for Beckhoff components there.

**Beckhoff Headquarters**

Beckhoff Automation GmbH & Co. KG

Huelshorstweg 20
33415 Verl
Germany

| | |
|---|---|
| Phone: | +49(0)5246/963-0 |
| Fax: | +49(0)5246/963-198 |
| e-mail: | info@beckhoff.com |

**Beckhoff Support**

Support offers you comprehensive technical assistance, helping you not only with the application of individual Beckhoff products, but also with other, wide-ranging services:

- support
- design, programming and commissioning of complex automation systems
- and extensive training program for Beckhoff system components

| | |
|---|---|
| Hotline: | +49(0)5246/963-157 |
| Fax: | +49(0)5246/963-9157 |
| e-mail: | support@beckhoff.com |

**Beckhoff Service**

The Beckhoff Service Center supports you in all matters of after-sales service:

- on-site service
- repair service
- spare parts service
- hotline service

| | |
|---|---|
| Hotline: | +49(0)5246/963-460 |
| Fax: | +49(0)5246/963-479 |
| e-mail: | service@beckhoff.com |

More Information:
**www.beckhoff.com/CX2500-M510**

Beckhoff Automation GmbH & Co. KG
Hülshorstweg 20
33415 Verl
Germany
Phone: +49 5246 9630
info@beckhoff.com
www.beckhoff.com