Documentation

EL6751

Master/Slave Terminal for CANopen

Version: 3.5
Date: 2019-07-12

BECKHOFF

Table of contents

1 Foreword

1.1 Notes on the documentation

Intended audience

This description is only intended for the use of trained specialists in control and automation engineering who are familiar with the applicable national standards.
It is essential that the documentation and the following notes and explanations are followed when installing and commissioning these 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® is 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 instructions

In this documentation the following instructions are used.
These instructions must be read carefully and followed without fail!

⚠ DANGER

Serious risk of injury!

Failure to follow this safety instruction directly endangers the life and health of persons.

⚠ WARNING

Risk of injury!

Failure to follow this safety instruction endangers the life and health of persons.

⚠ CAUTION

Personal injuries!

Failure to follow this safety instruction can lead to injuries to persons.

NOTE

Damage to environment/equipment or data loss

Failure to follow this instruction can lead to environmental damage, equipment damage or data loss.

Tip or pointer

This symbol indicates information that contributes to better understanding.

1.3 Documentation issue status

Version	Comment
3.5	• Update chapter „Object description“ • Structural update • Update revision status
3.4	• Update chapter „Object description“ • Structural update
3.3	• Update revision status • Structural update
3.2	• Update chapter „CANopen communication“ • Update chapter „Object description“ • Update revision status • Structural update
3.1	• "Technical data" chapter updated • Structural update
3.0	• Migration • Structural update
2.0	• "Technical data" chapter updated • Structural update
1.9	• Addenda chapter "Mounting and wiring"
1.8	• Addenda chapter "Mounting and wiring"
1.7	• Addenda firmware compatibility
1.6	• Additions to technical notes
1.5	• Additions to technical notes
1.4	• Chapter inserted "EtherCAT communication"
1.3	• Technical data corrected
1.2	• Supplementary notes CAN interface
1.1	• Addendum CAN Interface description
1.0	• Revision, technical data amended
0.1	• Preliminary version for internal use

1.4 Version identification of EtherCAT devices

Designation

A Beckhoff EtherCAT device has a 14-digit designation, made up of

- family key
- type
- version
- revision

Example	Family	Type	Version	Revision
EL3314-0000-0016	EL terminal (12 mm, non-pluggable connection level)	3314 (4-channel thermocouple terminal)	0000 (basic type)	0016
ES3602-0010-0017	ES terminal (12 mm, pluggable connection level)	3602 (2-channel voltage measurement)	0010 (high-precision version)	0017
CU2008-0000-0000	CU device	2008 (8-port fast ethernet switch)	0000 (basic type)	0000

Notes

- The elements mentioned above result in the **technical designation**. EL3314-0000-0016 is used in the example below.
- EL3314-0000 is the order identifier, in the case of “-0000” usually abbreviated to EL3314. “-0016” is the EtherCAT revision.
- The **order identifier** is made up of
  - family key (EL, EP, CU, ES, KL, CX, etc.)
  - type (3314)
  - version (-0000)
- The **revision** -0016 shows the technical progress, such as the extension of features with regard to the EtherCAT communication, and is managed by Beckhoff.
  In principle, a device with a higher revision can replace a device with a lower revision, unless specified otherwise, e.g. in the documentation.
  Associated and synonymous with each revision there is usually a description (ESI, EtherCAT Slave Information) in the form of an XML file, which is available for download from the Beckhoff web site.
  From 2014/01 the revision is shown on the outside of the IP20 terminals, see Fig. *“EL5021 EL terminal, standard IP20 IO device with batch number and revision ID (since 2014/01)”*.
- The type, version and revision are read as decimal numbers, even if they are technically saved in hexadecimal.

Identification number

Beckhoff EtherCAT devices from the different lines have different kinds of identification numbers:

Production lot/batch number/serial number/date code/D number

The serial number for Beckhoff IO devices is usually the 8-digit number printed on the device or on a sticker. The serial number indicates the configuration in delivery state and therefore refers to a whole production batch, without distinguishing the individual modules of a batch.

Structure of the serial number: **KK YY FF HH**

KK - week of production (CW, calendar week)
YY - year of production
FF - firmware version
HH - hardware version

Example with
Ser. no.: 12063A02: 12 - production week 12 06 - production year 2006 3A - firmware version 3A 02 - hardware version 02

Exceptions can occur in the **IP67 area**, where the following syntax can be used (see respective device documentation):

Syntax: D ww yy x y z u

D - prefix designation
ww - calendar week
yy - year
x - firmware version of the bus PCB

y - hardware version of the bus PCB
z - firmware version of the I/O PCB
u - hardware version of the I/O PCB

Example: D.22081501 calendar week 22 of the year 2008 firmware version of bus PCB: 1 hardware version of bus PCB: 5 firmware version of I/O PCB: 0 (no firmware necessary for this PCB) hardware version of I/O PCB: 1

Unique serial number/ID, ID number

In addition, in some series each individual module has its own unique serial number.

See also the further documentation in the area

- IP67: EtherCAT Box
- Safety: TwinSafe
- Terminals with factory calibration certificate and other measuring terminals

Examples of markings

Fig. 1: EL5021 EL terminal, standard IP20 IO device with serial/ batch number and revision ID (since 2014/01)

Fig. 2: EK1100 EtherCAT coupler, standard IP20 IO device with serial/ batch number

Fig. 3: CU2016 switch with serial/ batch number

Fig. 4: EL3202-0020 with serial/ batch number 26131006 and unique ID-number 204418

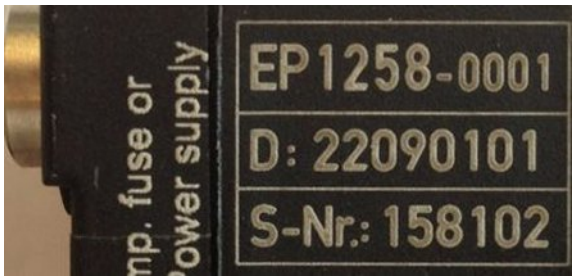


Fig. 5: EP1258-00001 IP67 EtherCAT Box with batch number/ date code 22090101 and unique serial number 158102

Fig. 6: EP1908-0002 IP67 EtherCAT Safety Box with batch number/ date code 071201FF and unique serial number 00346070

Fig. 7: EL2904 IP20 safety terminal with batch number/ date code 50110302 and unique serial number 00331701

Fig. 8: ELM3604-0002 terminal with unique ID number (QR code) 100001051 and serial/ batch number 44160201

1.4.1 Beckhoff Identification Code (BIC)

The Beckhoff Identification Code (BIC) is increasingly being applied to Beckhoff products to uniquely identify the product. The BIC is represented as a Data Matrix Code (DMC, code scheme ECC200), the content is based on the ANSI standard MH10.8.2-2016.

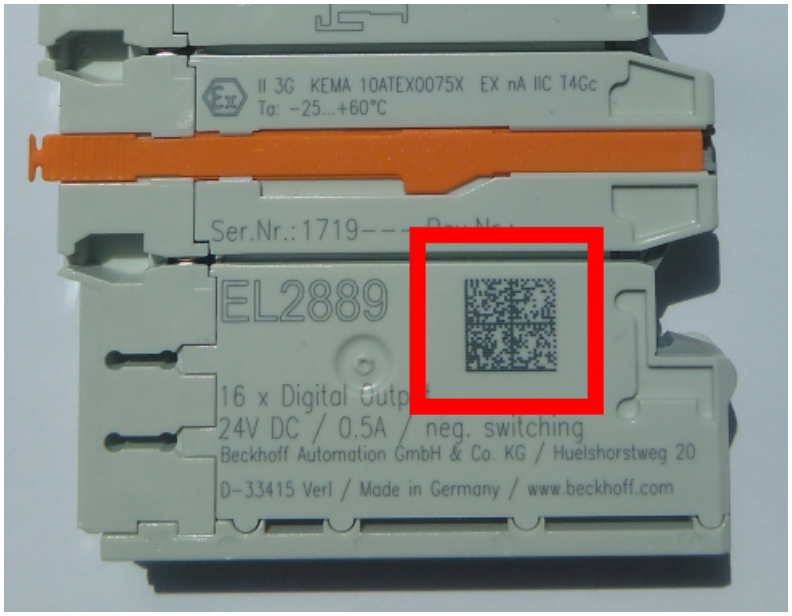


Fig. 9: BIC as data matrix code (DMC, code scheme ECC200)

The BIC will be introduced step by step across all product groups.

Depending on the product, it can be found in the following places:

- on the packaging unit
- directly on the product (if space suffices)
- on the packaging unit and the product

The BIC is machine-readable and contains information that can also be used by the customer for handling and product management.

Each piece of information can be uniquely identified using the so-called data identifier (ANSI MH10.8.2-2016). The data identifier is followed by a character string. Both together have a maximum length according to the table below. If the information is shorter, it shall be replaced by spaces. The data under positions 1-4 are always available.

The following information is contained:

Item no.	Type of information	Explanation	Data identifier	Number of digits incl. data identifier	Example
1	Beckhoff order number	**Beckhoff order number**	1P	8	1P072222
2	Beckhoff Traceability Number (BTN)	**Unique serial number, see note below**	S	12	SBTNk4p562d7
3	Article description	**Beckhoff article description, e.g. EL1008**	1K	32	1KEL1809
4	Quantity	**Quantity in packaging unit, e.g. 1, 10, etc.**	Q	6	Q1
5	Batch number	Optional: Year and week of production	2P	14	2P401503180016
6	ID/serial number	Optional: Present-day serial number system, e.g. with safety products	51S	12	51S678294104
7	Variant number	Optional: Product variant number on the basis of standard products	30P	32	30PF971 , 2*K183
...					

Further types of information and data identifiers are used by Beckhoff and serve internal processes.

Structure of the BIC

Example of composite information from items 1 - 4 and 6. The data identifiers are marked in red for better display:

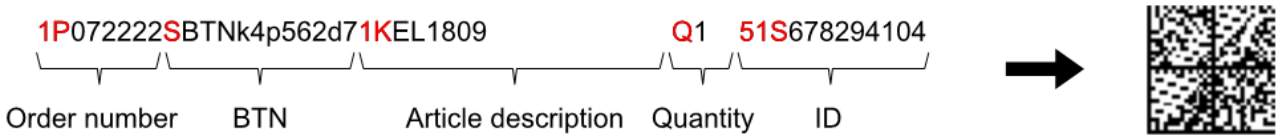


Fig. 10: Structure of the BIC

BTN

An important component of the BIC is the Beckhoff Traceability Number (BTN, item no. 2). The BTN is a unique serial number consisting of eight characters that will replace all other serial number systems at Beckhoff in the long term (e.g. batch designations on IO components, previous serial number range for safety products, etc.). The BTN will also be introduced step by step, so it may happen that the BTN is not yet coded in the BIC

Notice

This information has been carefully prepared. However, the procedure described is constantly being further developed. We reserve the right to revise and change procedures and documentation at any time and without prior notice. No claims for changes can be made from the information, illustrations and descriptions in this information.

2 Product overview

2.1 Introduction

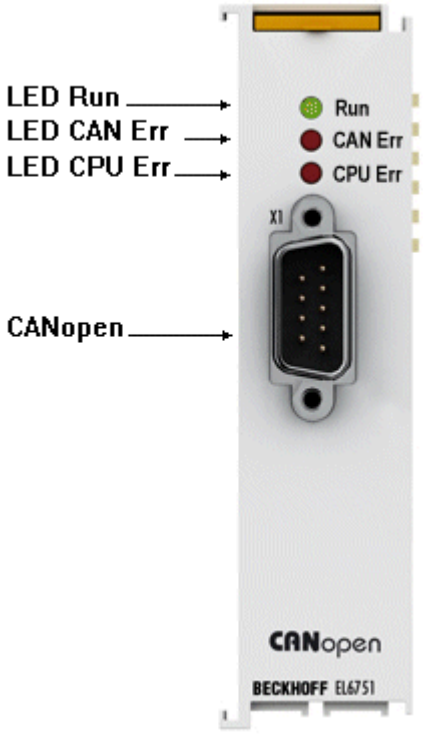


Fig. 11: EL6751

Master and slave terminals for CANopen

The master and slave terminals for CANopen correspond to the FC5101 PCI card from Beckhoff. Thanks to the connection via Ethernet, no PCI slots are required in the PC. Within an EtherCAT terminal network, it enables the integration of any CANopen devices. In addition, general CAN messages can be sent or received – without having to bother with CAN frames in the applications program. The EL6751 is alternatively available in a master or slave version and has a powerful protocol implementation with many features:

- All CANopen PDO communication types are supported: event-controlled, time-controlled (event timer), synchronous.
- Synchronization with the PC controller's task cycle.
- SYNC cycle with quartz precision for drive synchronization, zero cumulative jitter.
- Parameter communication (SDO) at start-up and at runtime.
- Emergency message handling, guarding and heartbeat.
- Powerful parameter and diagnostics interfaces.
- Online bus load display

Quick links

- EL6751 - CANopen master terminal [▶ 76]
- EL6751-0010 - CANopen slave terminal [▶ 82]

2.2 Technical data

Technical data	EL6751-0000	EL6751-0010
Bus system	CANopen	
Version	Master	Slave
Number of fieldbus channels	1	
Data transfer rate	10, 20, 50, 100, 125, 250, 500, 800 or 1000 kbaud	
Bus interface	D-Sub 9-pole connector according to CANopen specification, galvanically uncoupled	
Bus devices	maximum 127 slaves	
Communication	CANopen network master and CANopen manager	CANopen slave
Diagnostics	Status LEDs	
Power supply	via the E-bus	
Current consumption via E-bus	typ. 230 mA	
Electrical isolation	500 V (E-bus/CANopen)	
Configuration	with TwinCAT System Manager	
Weight	approx. 70 g	
Permissible ambient temperature range during operation	-25 °C ... +60 °C (extended temperature range)	
Permissible ambient temperature range during storage	-40 °C ... +85 °C	
Permissible relative humidity	95 %, no condensation	
Dimensions (W x H x D)	approx. 26 mm x 100 mm x 52 mm (width aligned: 23 mm)	
Mounting [▶ 18]	on 35 mm mounting rail conforms to EN 60715	
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	IP20	
Installation position	variable	
Approval	CE ATEX [▶ 25] cULus [▶ 26]	

2.3 CANopen Introduction

Fig. 12: CANopenLogo

CANopen is a widely used CAN application layer, developed by the CAN-in-Automation association (CiA, http://www.can-cia.org), and which has meanwhile been adopted for international standardization.

Device Model

CANopen consists of the protocol definitions (communication profile) and of the device profiles that standardize the data contents for the various device classes. Process data objects (PDO) [▶ 101] are used for fast communication of input and output data. The CANopen 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). There are, additionally, a few special objects (such as telegram types) for network management (NMT), synchronization, error messages and so on.

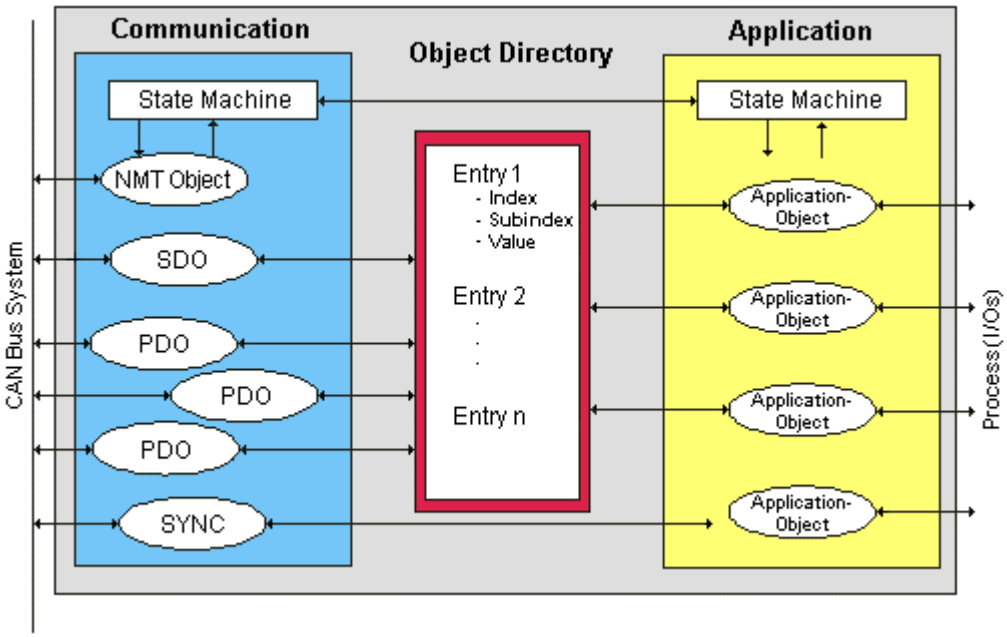


Fig. 13: CANopen Device Model

Communication Types

CANopen defines a number of communication classes for the input and output data (process data objects):

- Event driven [▶ 104]: 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 [▶ 104]: A SYNC telegram causes the modules to accept the output data that was previously received, and to send new input data.
- Requested (polled) [▶ 101]: A CAN data request telegram causes the modules to send their input data.

The desired communication type is set by the Transmission Type [▶ 101] parameter.

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.

Data transfer rates

Nine transmission rates [▶ 119] from 10 kbaud up to 1 Mbaud are available for different bus lengths. The effective utilization of the bus bandwidth allows CANopen to achieve short system reaction times at relatively low data rates.

Topology

CAN is based on a linear topology [▶ 27]. The number of devices participating in each network is logically limited by CANopen to 128, but physically the present generation of drivers allows up to 64 nodes in one network segment. The maximum possible size of the network for any particular data rate is limited by the signal propagation delay required on the bus medium. For 1 Mbaud, for instance, the network may extend 25 m, whereas at 50 kbaud the network may reach up to 1000 m. At low data rates the size of the network can be increased by repeaters, which also allow the construction of tree structures.

Bus access procedures

CAN utilizes the Carrier Sense Multiple Access (CSMA) procedure, i.e. all participating devices have the same right of access to the bus and may access it 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 assigns bus bandwidth to the messages in the sequence of their priority. At the end of the arbitration phase only one bus device occupies the bus, collisions are avoided and the bandwidth is optimally exploited.

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 (http://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 (http://www.can-cia.org).

3 Mounting and wiring

3.1 Recommended mounting rails

Terminal Modules und EtherCAT Modules of KMxxxx and EMxxxx series, same as the terminals of the EL66xx and EL67xx series can be snapped onto the following recommended mounting rails:

DIN Rail TH 35-7.5 with 1 mm material thickness (according to EN 60715)

DIN Rail TH 35-15 with 1,5 mm material thickness

Pay attention to the material thickness of the DIN Rail

Terminal Modules und EtherCAT Modules of KMxxxx and EMxxxx series, same as the terminals of the EL66xx and EL67xx series does not fit to the DIN Rail TH 35-15 with 2,2 to 2,5 mm material thickness (according to EN 60715)!

3.2 Mounting and demounting - terminals with traction lever unlocking

The terminal modules are fastened to the assembly surface with the aid of a 35 mm mounting rail (e.g. mounting rail TH 35-15).

Fixing of mounting rails

The locking mechanism of the terminals and couplers extends to the profile of the mounting rail. At the installation, the locking mechanism of the components must not come into conflict with the fixing bolts of the mounting rail. To mount the recommended mounting rails under the terminals and couplers, you should use flat mounting connections (e.g. countersunk screws or blind rivets).

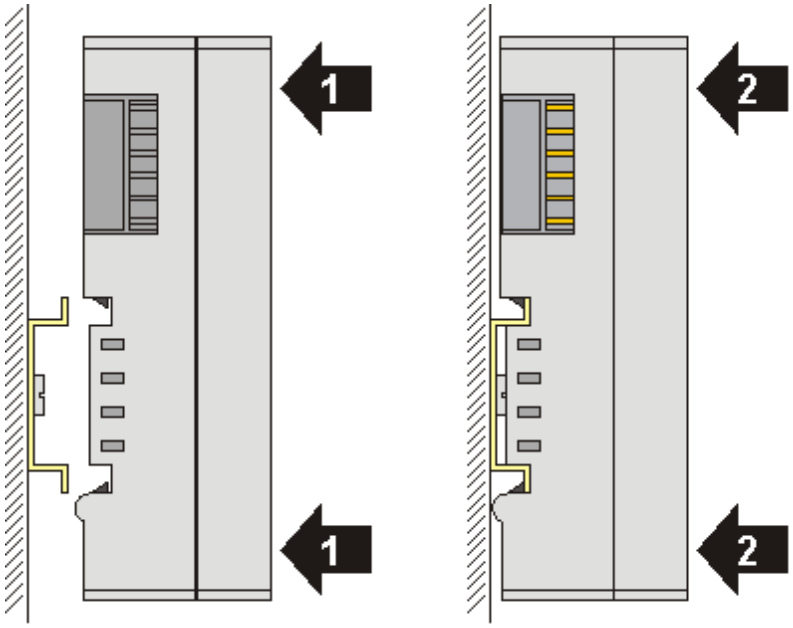
⚠ WARNING

Risk of electric shock and damage of device!

Bring the bus terminal system into a safe, powered down state before starting installation, disassembly or wiring of the Bus Terminals!

Mounting

- Fit the mounting rail to the planned assembly location.

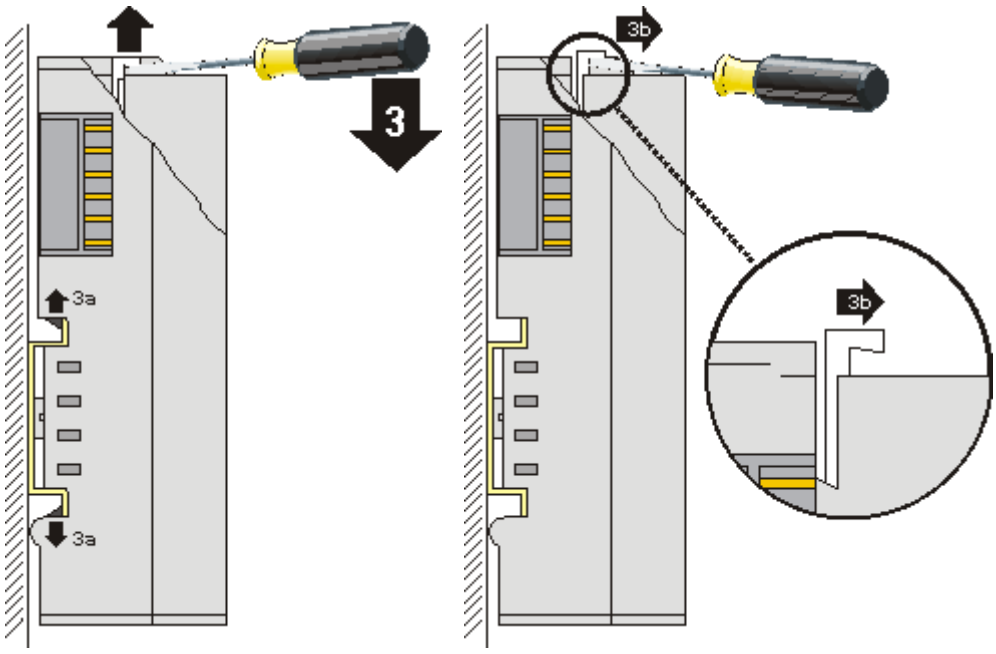


and press (1) the terminal module against the mounting rail until it latches in place on the mounting rail (2).

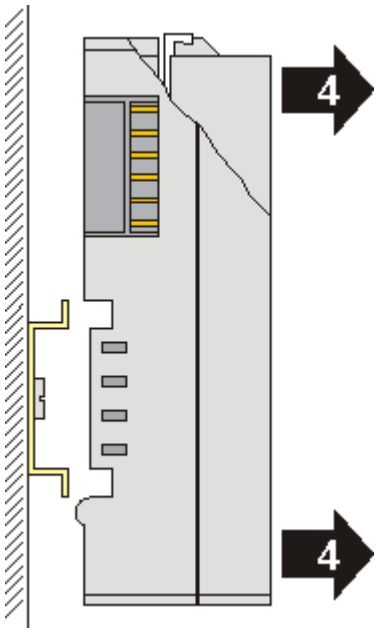
- Attach the cables.

Demounting

- Remove all the cables. Thanks to the KM/EM connector, it is not necessary to remove all the cables separately for this, but for each KM/EM connector simply undo 2 screws so that you can pull them off (fixed wiring)!
- Lever the unlatching hook on the left-hand side of the terminal module upwards with a screwdriver (3). As you do this
  - an internal mechanism pulls the two latching lugs (3a) from the top hat rail back into the terminal module,
  - the unlatching hook moves forwards (3b) and engages



- In the case 32 and 64 channel terminal modules (KMxxx4 and KMxxx8 or EMxxx4 and EMxxx8) you now lever the second unlatching hook on the right-hand side of the terminal module upwards in the same way.
- Pull (4) the terminal module away from the mounting surface.



3.3 Mounting and demounting - terminals with front unlocking

The terminal modules are fastened to the assembly surface with the aid of a 35 mm mounting rail (e.g. mounting rail TH 35-15).

Fixing of mounting rails

The locking mechanism of the terminals and couplers extends to the profile of the mounting rail. At the installation, the locking mechanism of the components must not come into conflict with the fixing bolts of the mounting rail. To mount the recommended mounting rails under the terminals and couplers, you should use flat mounting connections (e.g. countersunk screws or blind rivets).

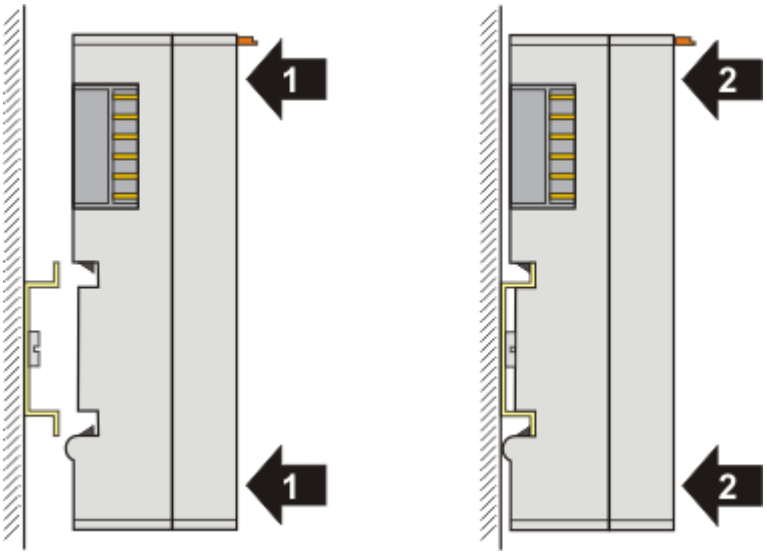
⚠ WARNING

Risk of electric shock and damage of device!

Bring the bus terminal system into a safe, powered down state before starting installation, disassembly or wiring of the Bus Terminals!

Mounting

- Fit the mounting rail to the planned assembly location.

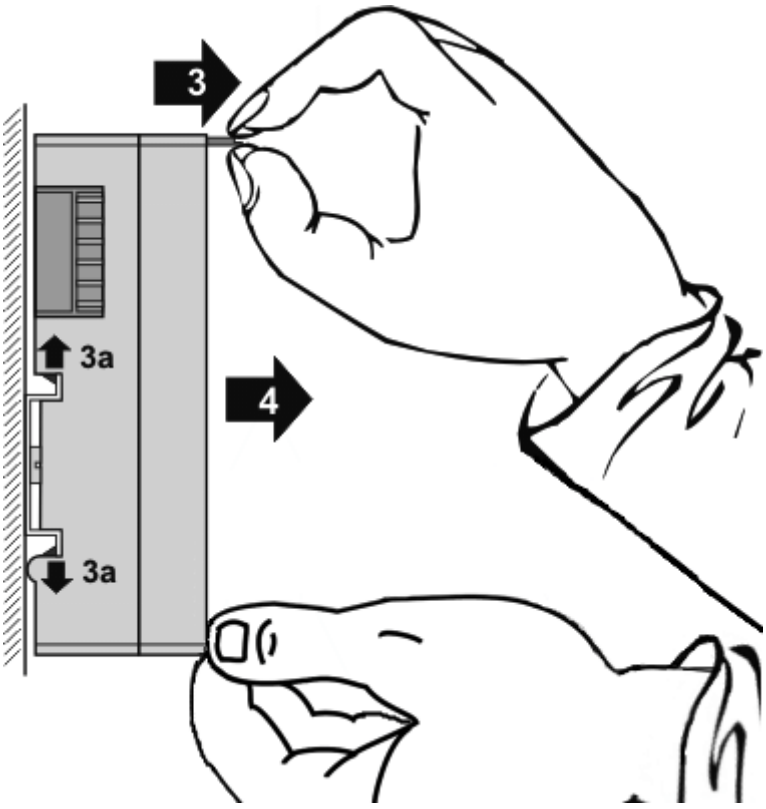


and press (1) the terminal module against the mounting rail until it latches in place on the mounting rail (2).

- Attach the cables.

Demounting

- Remove all the cables.
- Lever the unlatching hook back with thumb and forefinger (3). An internal mechanism pulls the two latching lugs (3a) from the top hat rail back into the terminal module.



- Pull (4) the terminal module away from the mounting surface.
  Avoid canting of the module; you should stabilize the module with the other hand, if required.

3.4 Installation positions

NOTE

Constraints regarding installation position and operating temperature range

Please refer to the technical data for a terminal to ascertain whether any restrictions regarding the installation position and/or the operating temperature range have been specified. When installing high power dissipation terminals ensure that an adequate spacing is maintained between other components above and below the terminal in order to guarantee adequate ventilation!

Optimum installation position (standard)

The optimum installation position requires the mounting rail to be installed horizontally and the connection surfaces of the EL/KL terminals to face forward (see Fig. *"Recommended distances for standard installation position"*). The terminals are ventilated from below, which enables optimum cooling of the electronics through convection. "From below" is relative to the acceleration of gravity.

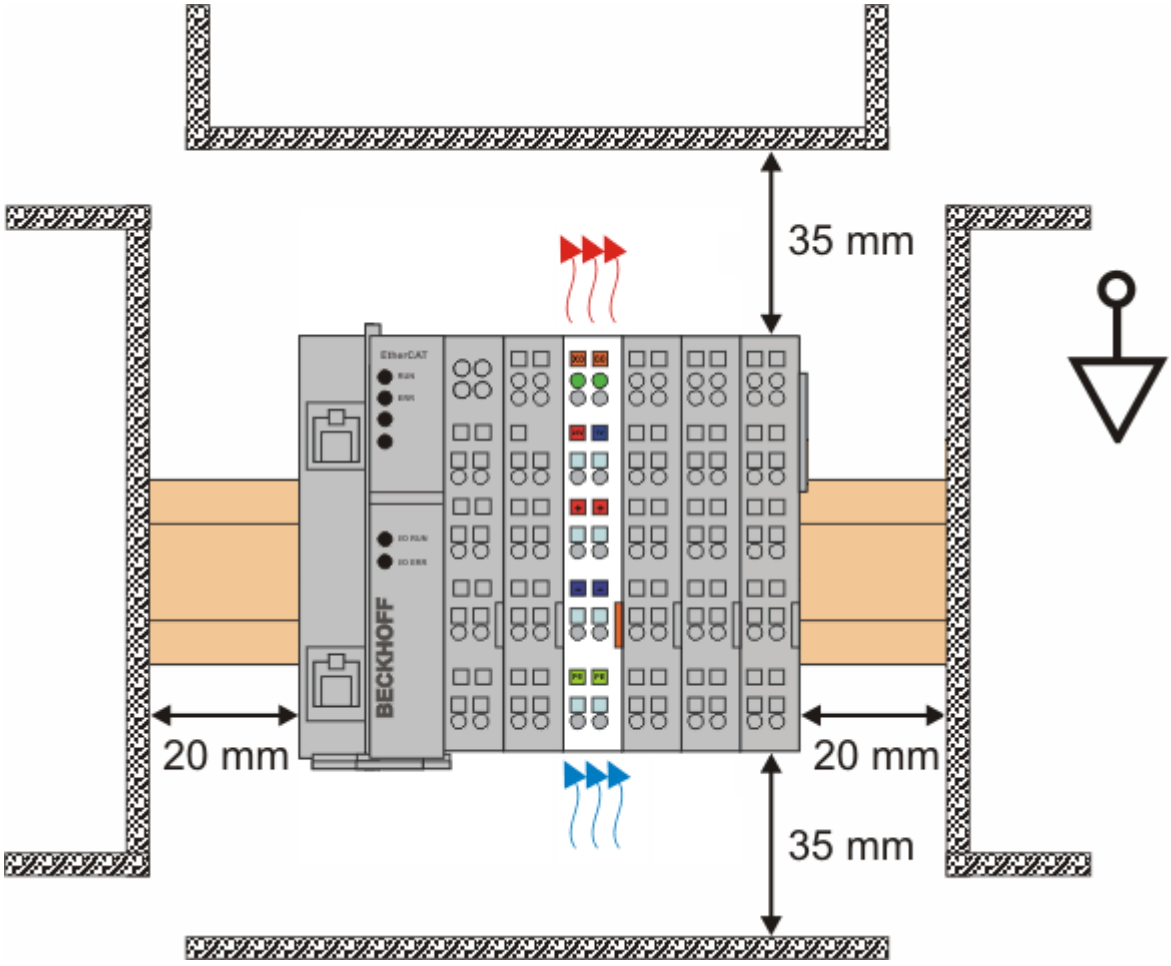


Fig. 14: Recommended distances for standard installation position

Compliance with the distances shown in Fig. *"Recommended distances for standard installation position"* is recommended.

Other installation positions

All other installation positions are characterized by different spatial arrangement of the mounting rail - see Fig *"Other installation positions"*.

The minimum distances to ambient specified above also apply to these installation positions.

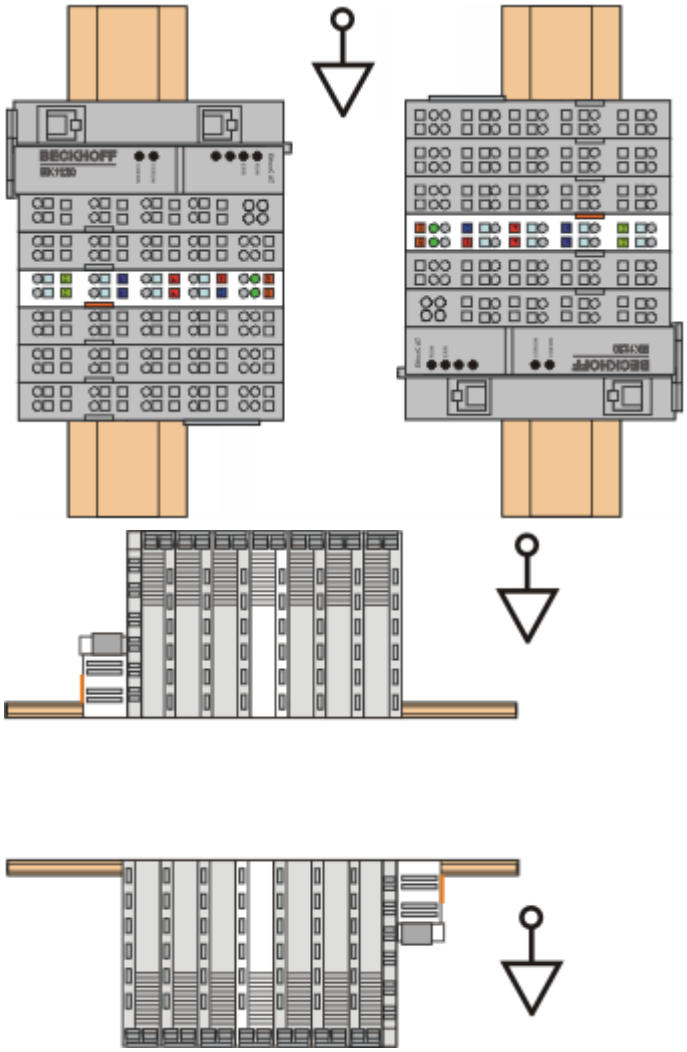


Fig. 15: Other installation positions

3.5 Positioning of passive Terminals

Hint for positioning of passive terminals in the bus terminal block

EtherCAT Terminals (ELxxxx / ESxxxx), which do not take an active part in data transfer within the bus terminal block are so called passive terminals. The passive terminals have no current consumption out of the E-Bus.
To ensure an optimal data transfer, you must not directly string together more than 2 passive terminals!

Examples for positioning of passive terminals (highlighted)

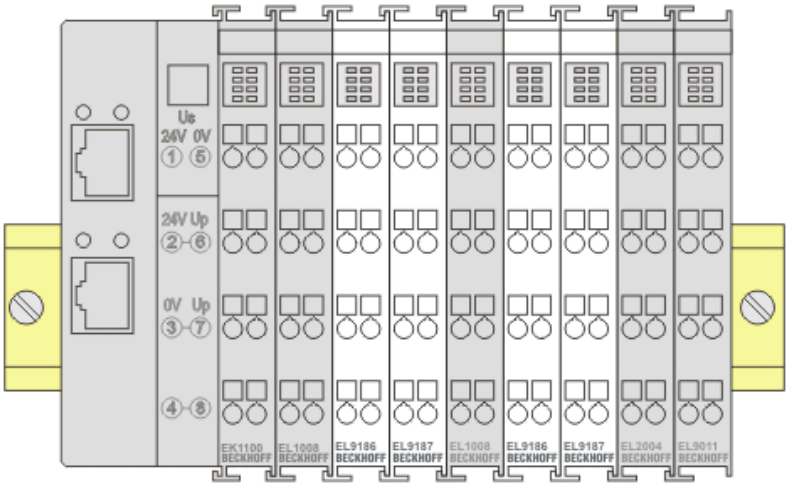


Fig. 16: Correct positioning

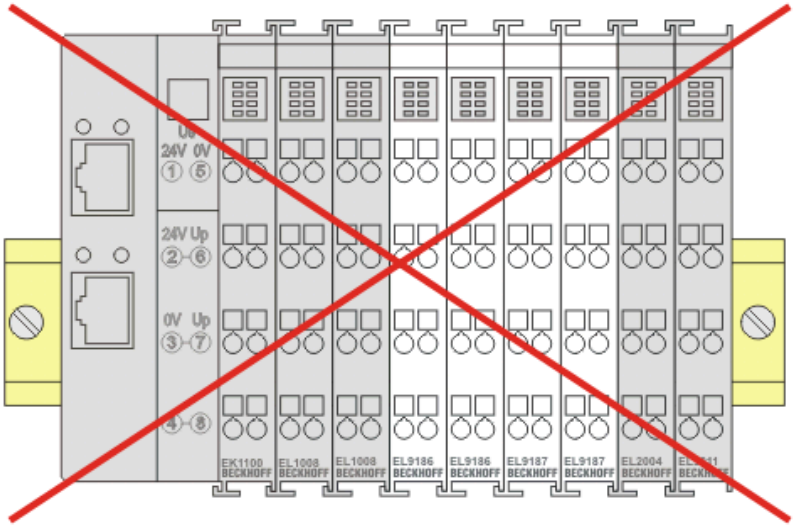


Fig. 17: Incorrect positioning

3.6 ATEX - Special conditions (extended temperature range)

⚠ WARNING

Observe the special conditions for the intended use of Beckhoff fieldbus components with extended temperature range (ET) in potentially explosive areas (directive 94/9/EU)!

- The certified components are to be installed in a suitable housing that guarantees a protection class of at least IP54 in accordance with EN 60529! The environmental conditions during use are thereby to be taken into account!
- If the temperatures during rated operation are higher than 70°C at the feed-in points of cables, lines or pipes, or higher than 80°C at the wire branching points, then cables must be selected whose temperature data correspond to the actual measured temperature values!
- Observe the permissible ambient temperature range of -25 to 60°C for the use of Beckhoff fieldbus components with extended temperature range (ET) in potentially explosive areas!
- Measures must be taken to protect against the rated operating voltage being exceeded by more than 40% due to short-term interference voltages!
- The individual terminals may only be unplugged or removed from the Bus Terminal system if the supply voltage has been switched off or if a non-explosive atmosphere is ensured!
- The connections of the certified components may only be connected or disconnected if the supply voltage has been switched off or if a non-explosive atmosphere is ensured!
- The fuses of the KL92xx/EL92xx power feed terminals may only be exchanged if the supply voltage has been switched off or if a non-explosive atmosphere is ensured!
- Address selectors and ID switches may only be adjusted if the supply voltage has been switched off or if a non-explosive atmosphere is ensured!

Standards

The fundamental health and safety requirements are fulfilled by compliance with the following standards:

- EN 60079-0:2012+A11:2013
- EN 60079-15:2010

Marking

The Beckhoff fieldbus components with extended temperature range (ET) certified for potentially explosive areas bear the following marking:

II 3G KEMA 10ATEX0075 X Ex nA IIC T4 Gc Ta: -25 … 60°C

or

II 3G KEMA 10ATEX0075 X Ex nC IIC T4 Gc Ta: -25 … 60°C

3.7 ATEX Documentation

Notes about operation of the Beckhoff terminal systems in potentially explosive areas (ATEX)

Pay also attention to the continuative documentation

Notes about operation of the Beckhoff terminal systems in potentially explosive areas (ATEX)

that is available in the download area of the Beckhoff homepage http:\\www.beckhoff.com!

3.8 UL notice

c UL us	**Application** Beckhoff EtherCAT modules are intended for use with Beckhoff's UL Listed EtherCAT System only.
c UL us	**Examination** For cULus examination, the Beckhoff I/O System has only been investigated for risk of fire and electrical shock (in accordance with UL508 and CSA C22.2 No. 142).
c UL us	**For devices with Ethernet connectors** Not for connection to telecommunication circuits.

Basic principles

UL certification according to UL508. Devices with this kind of certification are marked by this sign:

c UL us

3.9 CANopen cabling

Notes related to checking the CAN wiring can be found in the Trouble Shooting [▶ 173] section.

3.9.1 CAN topology

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!

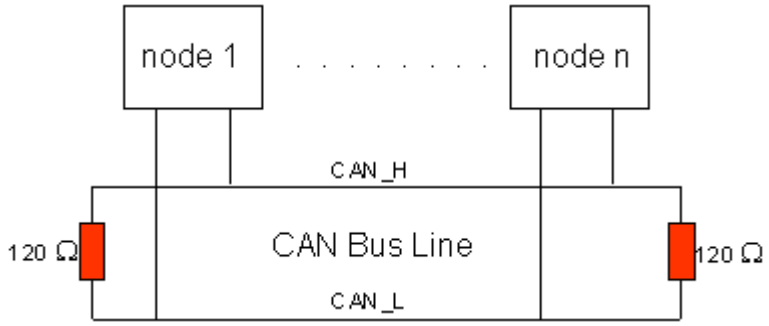


Fig. 18: Termination of the bus with a 120 Ohm termination resistor

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.

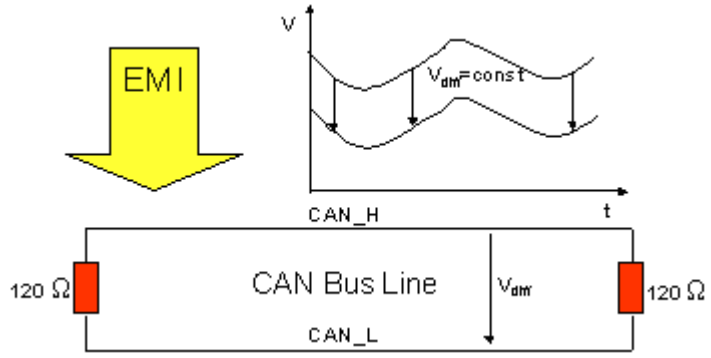


Fig. 19: Insensitivity to incoming interference

3.9.2 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 40 m 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.

3.9.3 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 [▶ 119] 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	< 1 m	< 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.

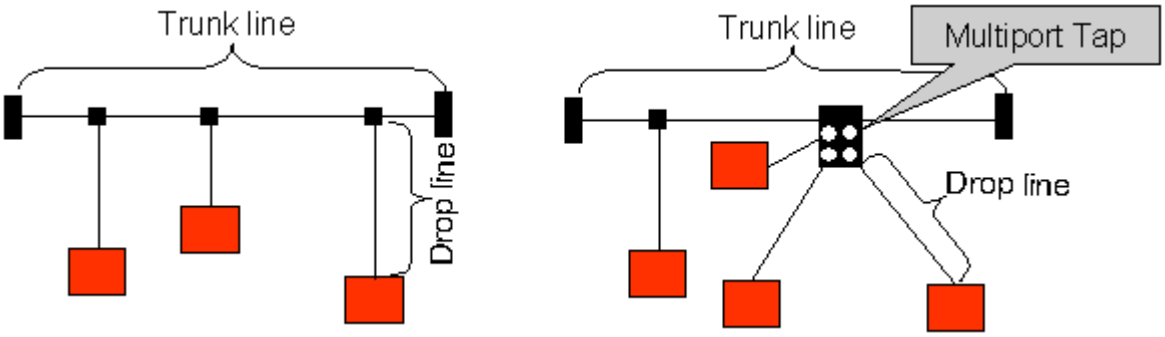


Fig. 20: Sample topology of drop lines

3.9.4 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

3.9.5 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 (60 kHz): 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)

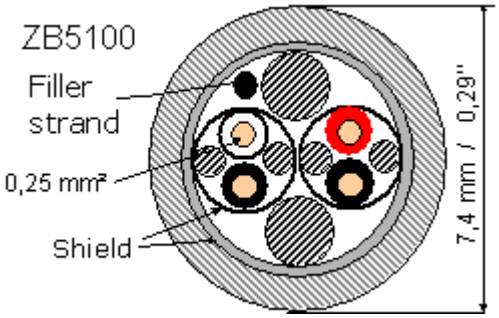


Fig. 21: Structure of CAN cable ZB5100

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

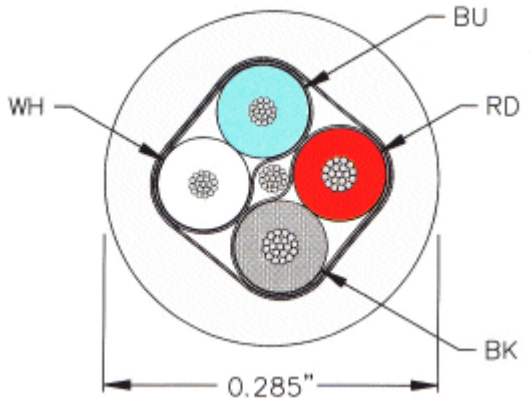


Fig. 22: Structure of CAN/DeviceNet cable ZB5200

3.9.6 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.

Notes related to checking the CAN wiring can be found in the Trouble Shooting [▶ 173] section.

3.9.7 Cable colors

Suggested method of using the Beckhoff CAN cable on Bus Terminal and Fieldbus Box:

BK51x0 pin PIN BX5100 (X510)	Pin BK5151 CX8050, CX8051, CXxxxx-B510/M510	Fieldbus Box pin	Pin FC51xx	Function	ZB5100 cable color	ZB5200 cable color
1	3	3	3	CAN Ground	**black/** (red)	**black**
2	2	5	2	CAN Low	**black**	**blue**
3	5	1	5	Shield	Filler strand	Filler strand
4	7	4	7	CAN high	**white**	**white**
5	9	2	9	not used	**(red)**	**(red)**

3.9.8 BK5151, FC51xx, CX with CAN interface and EL6751: D-sub, 9 pin

The CANbus cable is connected to the FC51x1, FC51x2 CANopen cards and in the case of the EL6751 CANopen master/slave terminal via 9-pin Sub-D sockets with the following pin assignment.

Pin	Assignment
2	CAN low (CAN-)
3	CAN ground (internally connected to pin 6)
6	CAN ground (internally connected to pin 3)
7	CAN high (CAN+)

The unlisted pins are not connected.
The mounting rail contact spring and the plug shield are connected together.

Note: an auxiliary voltage of up to 30 V_{DC} may be connected to pin 9. Some CAN devices use this to supply the transceiver.

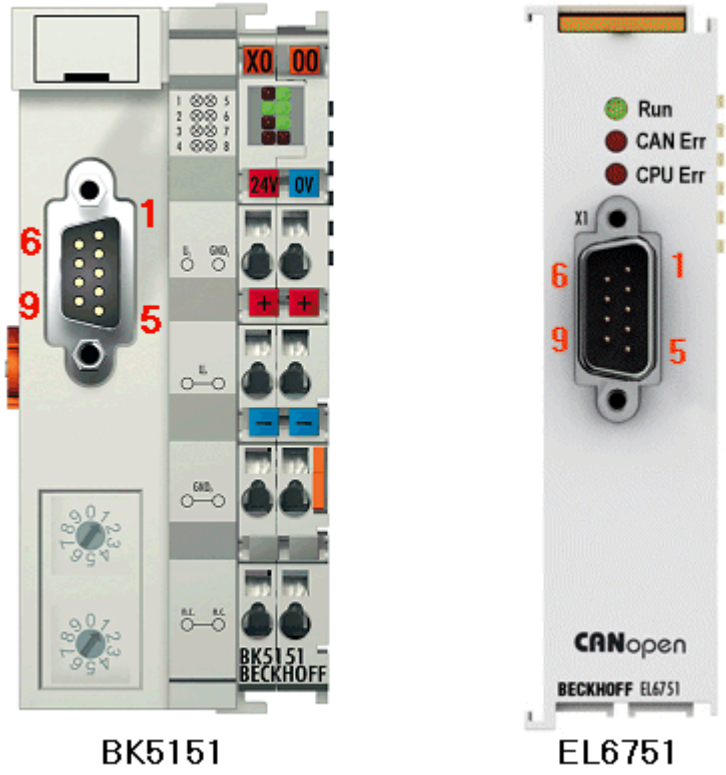


Fig. 23: BK5151, EL6751 pin assignment

FC51x2:

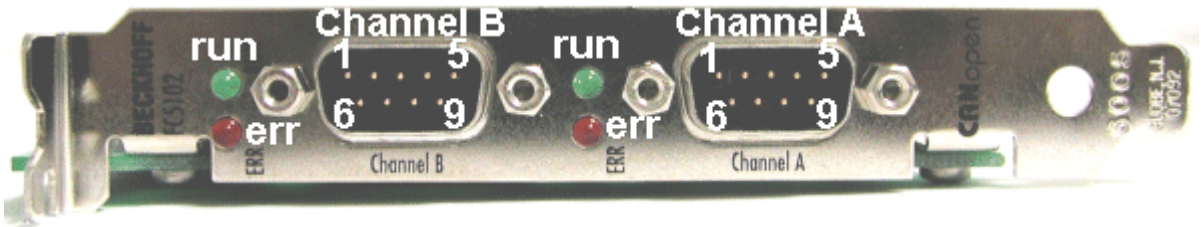


Fig. 24: FC51x2

3.9.9 BK51x0/BX5100: 5-pin open style connector

The BK51x0/BX5100 (X510) Bus Couplers have a recessed front surface on the left hand side with a five pin connector.
The supplied CANopen socket can be inserted here.

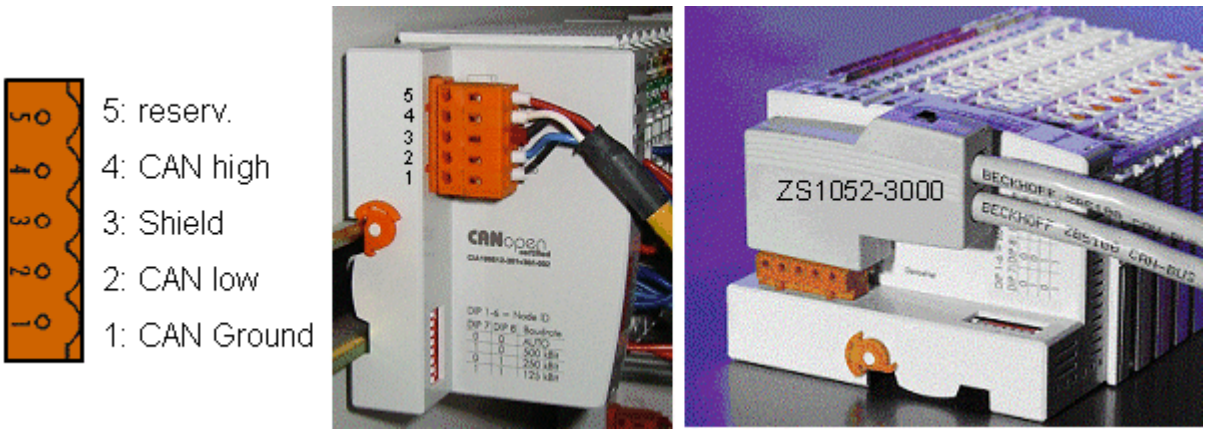


Fig. 25: BK51x0/BX5100 socket assignment

The left figure shows the socket in the BK51x0/BX5100 Bus Coupler. Pin 5 is the connection strip's top most pin. Pin 5 is not used. Pin 4 is the CAN high connection, pin 2 is the CAN low connection, and the screen is connected to pin 3 (which is connected to the mounting rail via an R/C network). CAN-GND can optionally be connected to pin 1. If all the CAN ground pins are connected, this provides a common reference potential for the CAN transceivers in the network. It is recommended that the CAN GND be connected to earth at one location, so that the common CAN reference potential is close to the supply potential. Since the CANopen BK51X0/BX5100 Bus Couplers provide full electrical isolation of the bus connection, it may in appropriate cases be possible to omit wiring up the CAN ground.

ZS1052-3000 Bus Interface Connector

The ZS1052-3000 CAN Interface Connector can be used as an alternative to the supplied connector. This makes the wiring significantly easier. There are separate terminals for incoming and outgoing leads and a large area of the screen is connected via the strain relief. The integrated terminating resistor can be switched externally. When it is switched on, the outgoing bus lead is electrically isolated - this allows rapid wiring fault location and guarantees that no more than two resistors are active in the network.

3.9.10 LC5100: Bus connection via spring-loaded terminals

In the low cost LC5100 Coupler, the CAN wires are connected directly to the contact points 1 (CAN-H, marked with C+) and 5 (CAN-L, marked with C-). The screen can optionally be connected to contact points 4 or 8, which are connected to the mounting rail via an R/C network.

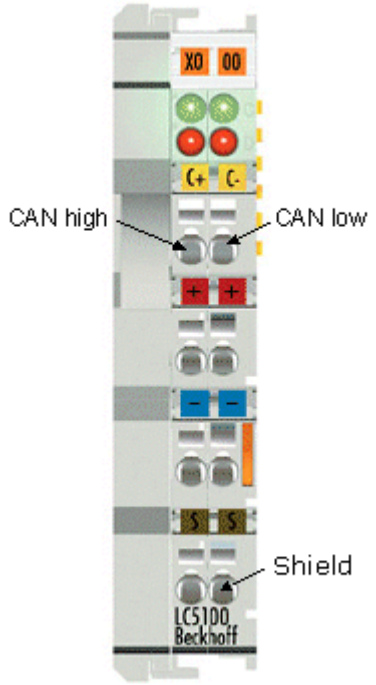


Fig. 26: LC5100

NOTE

Risk of device damage!

On account of the lack of electrical isolation, the CAN driver can be destroyed or damaged due to incorrect cabling. Always carry out the cabling in the switched-off condition.
First connect the power supply and then the CAN. Check the cabling and only then switch on the voltage.

3.9.11 Fieldbus Box: M12 CAN socket

The IPxxxx-B510, IL230x-B510 and IL230x-C510 Fieldbus Boxes are connected to the bus using 5-pin M12 plug-in connectors.

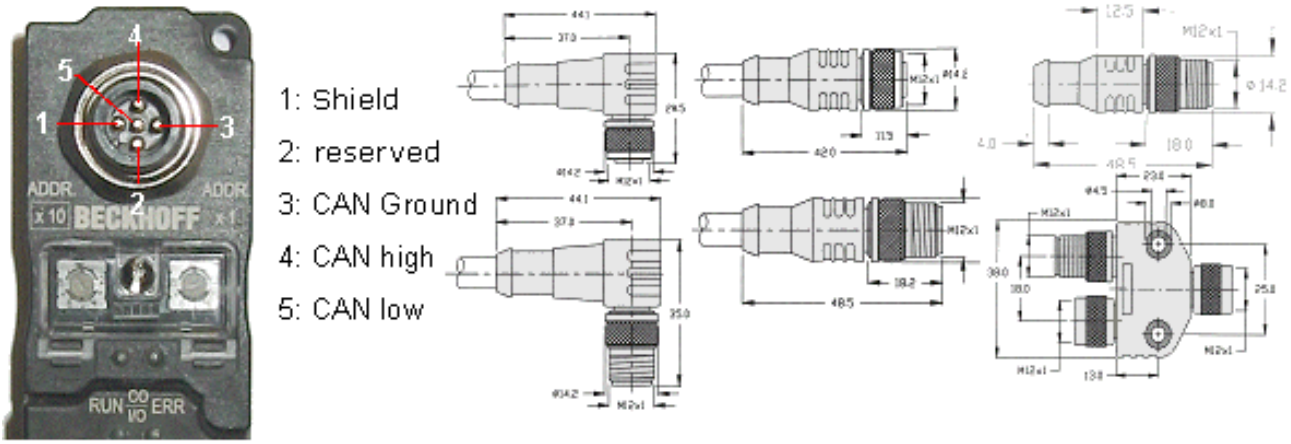


Fig. 27: Pin assignment: M12 plug, fieldbus box

Beckhoff offer plugs for field assembly, passive distributor's, terminating resistors and a wide range of pre-assembled cables for the Fieldbus Box system. Details be found in the catalogue, or under www.beckhoff.de.

4 Basics communication

4.1 EtherCAT basics

Please refer to the EtherCAT System Documentation for the EtherCAT fieldbus basics.

4.2 EtherCAT State Machine

The state of the EtherCAT slave is controlled via the EtherCAT State Machine (ESM). Depending upon the state, different functions are accessible or executable in the EtherCAT slave. Specific commands must be sent by the EtherCAT master to the device in each state, particularly during the bootup of the slave.

A distinction is made between the following states:

- Init
- Pre-Operational
- Safe-Operational and
- Operational
- Boot

The regular state of each EtherCAT slave after bootup is the OP state.

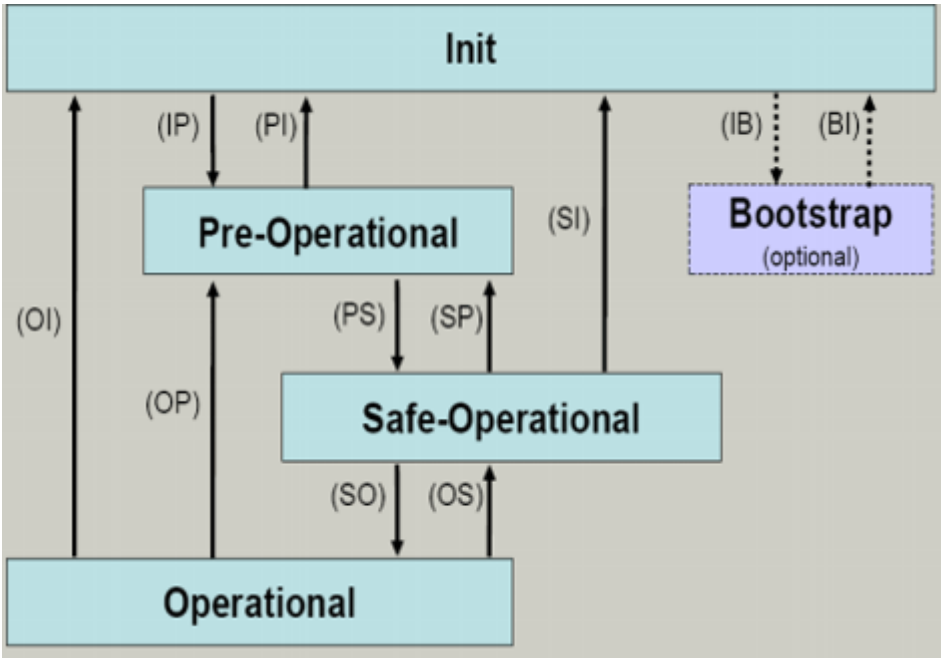


Fig. 28: States of the EtherCAT State Machine

Init

After switch-on the EtherCAT slave in the *Init* state. No mailbox or process data communication is possible. The EtherCAT master initializes sync manager channels 0 and 1 for mailbox communication.

Pre-Operational (Pre-Op)

During the transition between *Init* and *Pre-Op* the EtherCAT slave checks whether the mailbox was initialized correctly.

In *Pre-Op* state mailbox communication is possible, but not process data communication. The EtherCAT master initializes the sync manager channels for process data (from sync manager channel 2), the FMMU channels and, if the slave supports configurable mapping, PDO mapping or the sync manager PDO assignment. In this state the settings for the process data transfer and perhaps terminal-specific parameters that may differ from the default settings are also transferred.

Safe-Operational (Safe-Op)

During transition between *Pre-Op* and *Safe-Op* the EtherCAT slave checks whether the sync manager channels for process data communication and, if required, the distributed clocks settings are correct. Before it acknowledges the change of state, the EtherCAT slave copies current input data into the associated DP-RAM areas of the EtherCAT slave controller (ECSC).

In *Safe-Op* state mailbox and process data communication is possible, although the slave keeps its outputs in a safe state, while the input data are updated cyclically.

Outputs in SAFEOP state

The default set watchdog [▶ 35] monitoring sets the outputs of the module in a safe state - depending on the settings in SAFEOP and OP - e.g. in OFF state. If this is prevented by deactivation of the watchdog monitoring in the module, the outputs can be switched or set also in the SAFEOP state.

Operational (Op)

Before the EtherCAT master switches the EtherCAT slave from *Safe-Op* to *Op* it must transfer valid output data.

In the *Op* state the slave copies the output data of the masters to its outputs. Process data and mailbox communication is possible.

Boot

In the *Boot* state the slave firmware can be updated. The *Boot* state can only be reached via the *Init* state.

In the *Boot* state mailbox communication via the *file access over EtherCAT* (FoE) protocol is possible, but no other mailbox communication and no process data communication.

4.3 General notes for setting the watchdog

ELxxxx terminals are equipped with a safety feature (watchdog) that switches off the outputs after a specifiable time e.g. in the event of an interruption of the process data traffic, depending on the device and settings, e.g. in OFF state.

The EtherCAT slave controller (ESC) in the EL2xxx terminals features 2 watchdogs:

- SM watchdog (default: 100 ms)
- PDI watchdog (default: 100 ms)

SM watchdog (SyncManager Watchdog)

The SyncManager watchdog is reset after each successful EtherCAT process data communication with the terminal. If no EtherCAT process data communication takes place with the terminal for longer than the set and activated SM watchdog time, e.g. in the event of a line interruption, the watchdog is triggered and the outputs are set to FALSE. The OP state of the terminal is unaffected. The watchdog is only reset after a successful EtherCAT process data access. Set the monitoring time as described below.

The SyncManager watchdog monitors correct and timely process data communication with the ESC from the EtherCAT side.

PDI watchdog (Process Data Watchdog)

If no PDI communication with the EtherCAT slave controller (ESC) takes place for longer than the set and activated PDI watchdog time, this watchdog is triggered.
PDI (Process Data Interface) is the internal interface between the ESC and local processors in the EtherCAT slave, for example. The PDI watchdog can be used to monitor this communication for failure.

The PDI watchdog monitors correct and timely process data communication with the ESC from the application side.

The settings of the SM- and PDI-watchdog must be done for each slave separately in the TwinCAT System Manager.

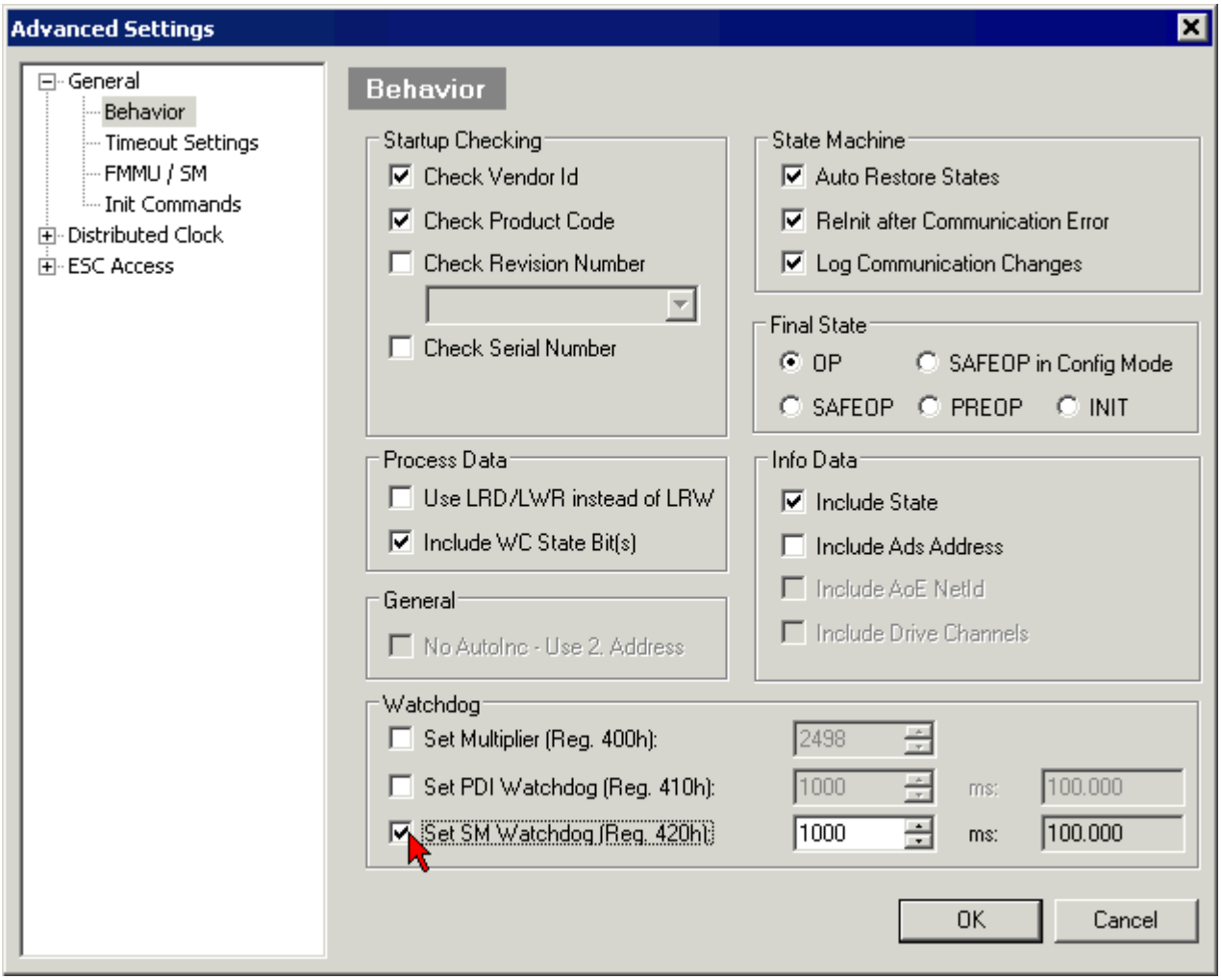


Fig. 29: *EtherCAT tab -> Advanced Settings -> Behavior -> Watchdog*

Notes:

- the multiplier is valid for both watchdogs.
- each watchdog has its own timer setting, the outcome of this in summary with the multiplier is a resulting time.
- Important: the multiplier/timer setting is only loaded into the slave at the start up, if the checkbox is activated.
  If the checkbox is not activated, nothing is downloaded and the ESC settings remain unchanged.

Multiplier

Multiplier

Both watchdogs receive their pulses from the local terminal cycle, divided by the watchdog multiplier:

1/25 MHz * (watchdog multiplier + 2) = 100 µs (for default setting of 2498 for the multiplier)

The standard setting of 1000 for the SM watchdog corresponds to a release time of 100 ms.

The value in multiplier + 2 corresponds to the number of basic 40 ns ticks representing a watchdog tick. The multiplier can be modified in order to adjust the watchdog time over a larger range.

Example "Set SM watchdog"

This checkbox enables manual setting of the watchdog times. If the outputs are set and the EtherCAT communication is interrupted, the SM watchdog is triggered after the set time and the outputs are erased. This setting can be used for adapting a terminal to a slower EtherCAT master or long cycle times. The default SM watchdog setting is 100 ms. The setting range is 0..65535. Together with a multiplier with a range of 1..65535 this covers a watchdog period between 0..~170 seconds.

Calculation

Multiplier = 2498 → watchdog base time = 1 / 25 MHz * (2498 + 2) = 0.0001 seconds = 100 µs
SM watchdog = 10000 → 10000 * 100 µs = 1 second watchdog monitoring time

⚠ CAUTION

Undefined state possible!

The function for switching off of the SM watchdog via SM watchdog = 0 is only implemented in terminals from version -0016. In previous versions this operating mode should not be used.

⚠ CAUTION

Damage of devices and undefined state possible!

If the SM watchdog is activated and a value of 0 is entered the watchdog switches off completely. This is the deactivation of the watchdog! Set outputs are NOT set in a safe state, if the communication is interrupted.

4.4 CoE Interface

General description

The CoE interface (CANopen over EtherCAT) is used for parameter management of EtherCAT devices. EtherCAT slaves or the EtherCAT master manage fixed (read only) or variable parameters which they require for operation, diagnostics or commissioning.

CoE parameters are arranged in a table hierarchy. In principle, the user has read access via the fieldbus. The EtherCAT master (TwinCAT System Manager) can access the local CoE lists of the slaves via EtherCAT in read or write mode, depending on the attributes.

Different CoE parameter types are possible, including string (text), integer numbers, Boolean values or larger byte fields. They can be used to describe a wide range of features. Examples of such parameters include manufacturer ID, serial number, process data settings, device name, calibration values for analog measurement or passwords.

The order is specified in 2 levels via hexadecimal numbering: (main)index, followed by subindex. The value ranges are

- Index: 0x0000 …0xFFFF ($0...65535_{dez}$)
- SubIndex: 0x00…0xFF ($0...255_{dez}$)

A parameter localized in this way is normally written as 0x8010:07, with preceding "x" to identify the hexadecimal numerical range and a colon between index and subindex.

The relevant ranges for EtherCAT fieldbus users are:

- 0x1000: This is where fixed identity information for the device is stored, including name, manufacturer, serial number etc., plus information about the current and available process data configurations.

- 0x8000: This is where the operational and functional parameters for all channels are stored, such as filter settings or output frequency.

Other important ranges are:

- 0x4000: In some EtherCAT devices the channel parameters are stored here (as an alternative to the 0x8000 range).
- 0x6000: Input PDOs ("input" from the perspective of the EtherCAT master)
- 0x7000: Output PDOs ("output" from the perspective of the EtherCAT master)

Availability

Not every EtherCAT device must have a CoE list. Simple I/O modules without dedicated processor usually have no variable parameters and therefore no CoE list.

If a device has a CoE list, it is shown in the TwinCAT System Manager as a separate tab with a listing of the elements:

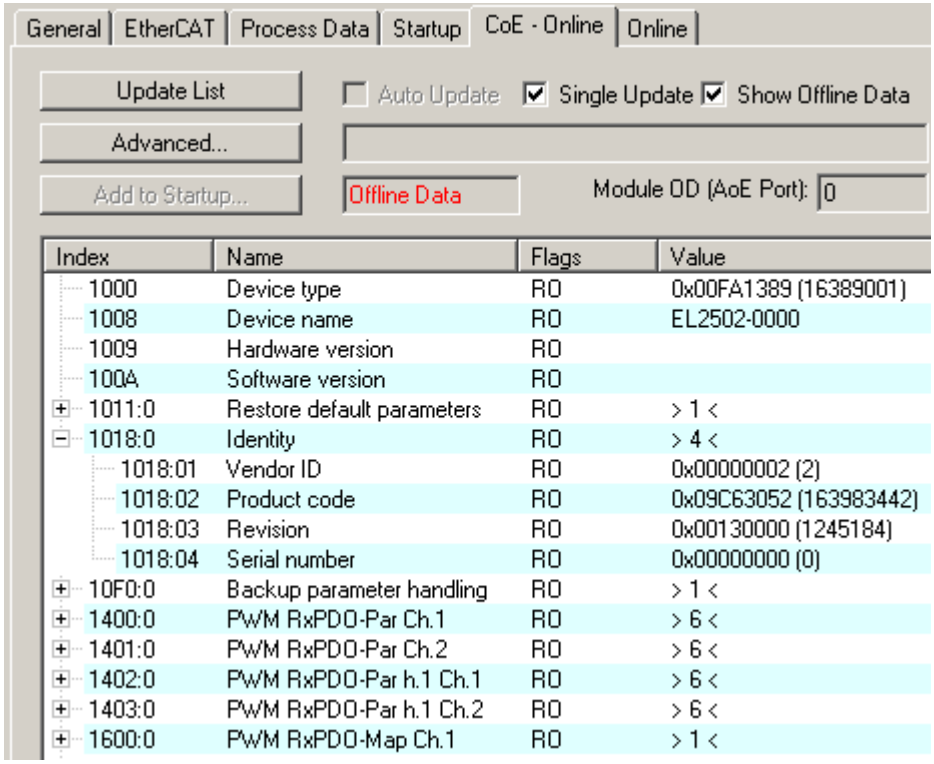


Fig. 30: *"CoE Online " tab*

The figure above shows the CoE objects available in device "EL2502", ranging from 0x1000 to 0x1600. The subindices for 0x1018 are expanded.

Data management and function "NoCoeStorage"

Some parameters, particularly the setting parameters of the slave, are configurable and writeable. This can be done in write or read mode

- via the System Manager (Fig. *"CoE Online " tab*) by clicking
  This is useful for commissioning of the system/slaves. Click on the row of the index to be parameterised and enter a value in the "SetValue" dialog.
- from the control system/PLC via ADS, e.g. through blocks from the TcEtherCAT.lib library
  This is recommended for modifications while the system is running or if no System Manager or operating staff are available.

Data management

If slave CoE parameters are modified online, Beckhoff devices store any changes in a fail-safe manner in the EEPROM, i.e. the modified CoE parameters are still available after a restart.
The situation may be different with other manufacturers.

An EEPROM is subject to a limited lifetime with respect to write operations. From typically 100,000 write operations onwards it can no longer be guaranteed that new (changed) data are reliably saved or are still readable. This is irrelevant for normal commissioning. However, if CoE parameters are continuously changed via ADS at machine runtime, it is quite possible for the lifetime limit to be reached. Support for the NoCoeStorage function, which suppresses the saving of changed CoE values, depends on the firmware version.
Please refer to the technical data in this documentation as to whether this applies to the respective device.

- If the function is supported: the function is activated by entering the code word 0x12345678 once in CoE 0xF008 and remains active as long as the code word is not changed. After switching the device on it is then inactive. Changed CoE values are not saved in the EEPROM and can thus be changed any number of times.
- Function is not supported: continuous changing of CoE values is not permissible in view of the lifetime limit.

Startup list

Changes in the local CoE list of the terminal are lost if the terminal is replaced. If a terminal is replaced with a new Beckhoff terminal, it will have the default settings. It is therefore advisable to link all changes in the CoE list of an EtherCAT slave with the Startup list of the slave, which is processed whenever the EtherCAT fieldbus is started. In this way a replacement EtherCAT slave can automatically be parameterized with the specifications of the user.

If EtherCAT slaves are used which are unable to store local CoE values permanently, the Startup list must be used.

Recommended approach for manual modification of CoE parameters

- Make the required change in the System Manager
  The values are stored locally in the EtherCAT slave
- If the value is to be stored permanently, enter it in the Startup list.
  The order of the Startup entries is usually irrelevant.

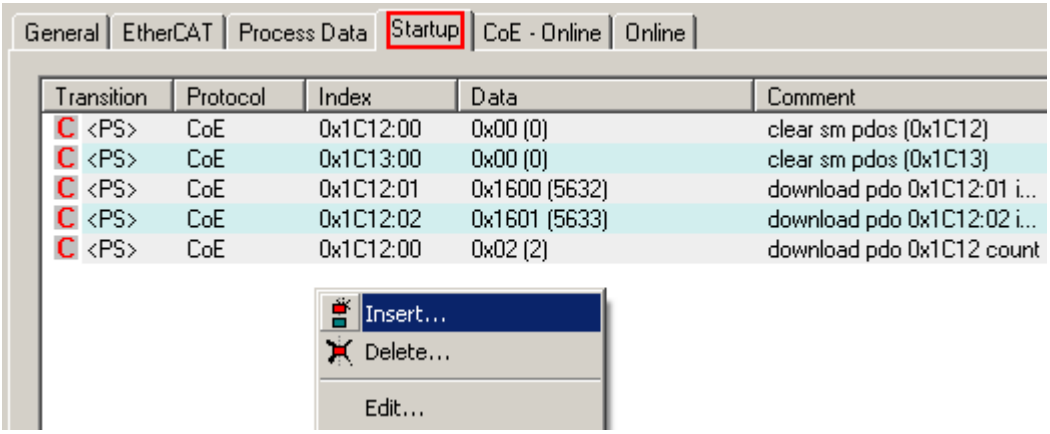


Fig. 31: *Startup list in the TwinCAT System Manager*

The Startup list may already contain values that were configured by the System Manager based on the ESI specifications. Additional application-specific entries can be created.

Online/offline list

While working with the TwinCAT System Manager, a distinction has to be made whether the EtherCAT device is "available", i.e. switched on and linked via EtherCAT and therefore **online**, or whether a configuration is created **offline** without connected slaves.

In both cases a CoE list as shown in Fig. *"'CoE online' tab"* is displayed. The connectivity is shown as offline/ online.

- If the slave is offline
  - The offline list from the ESI file is displayed. In this case modifications are not meaningful or possible.
  - The configured status is shown under Identity.
  - No firmware or hardware version is displayed, since these are features of the physical device.
  - **Offline** is shown in red.

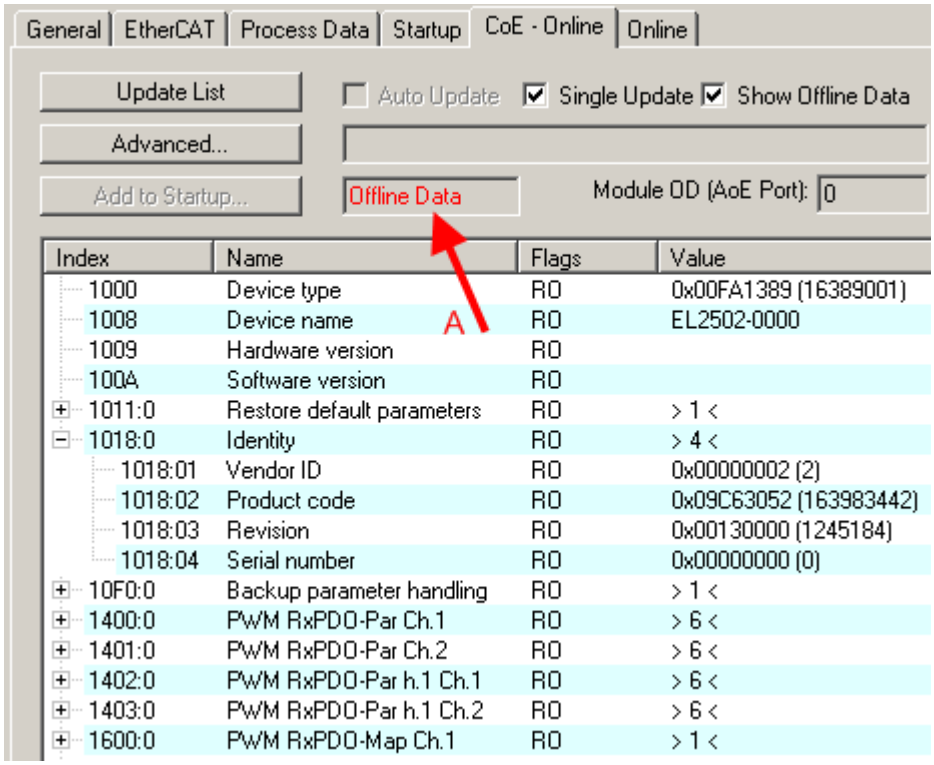


Fig. 32: *Offline list*

- If the slave is online
  - The actual current slave list is read. This may take several seconds, depending on the size and cycle time.
  - The actual identity is displayed
  - The firmware and hardware version of the equipment according to the electronic information is displayed
  - **Online** is shown in green.

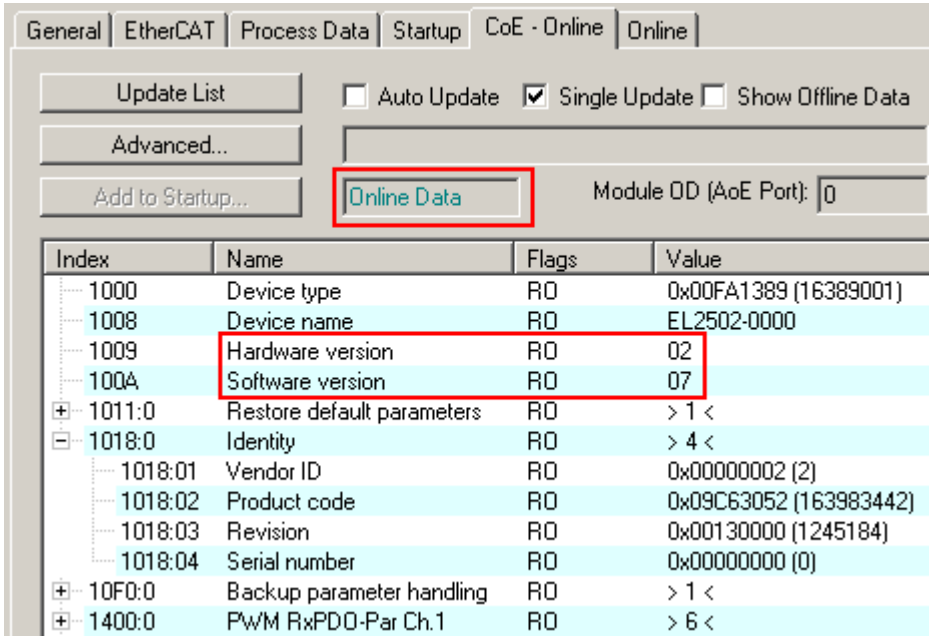


Fig. 33: *Online list*

Channel-based order

The CoE list is available in EtherCAT devices that usually feature several functionally equivalent channels. For example, a 4-channel analog 0..10 V input terminal also has 4 logical channels and therefore 4 identical sets of parameter data for the channels. In order to avoid having to list each channel in the documentation, the placeholder "n" tends to be used for the individual channel numbers.

In the CoE system 16 indices, each with 255 subindices, are generally sufficient for representing all channel parameters. The channel-based order is therefore arranged in $16_{dec}/10_{hex}$ steps. The parameter range 0x8000 exemplifies this:

- Channel 0: parameter range 0x8000:00 ... 0x800F:255
- Channel 1: parameter range 0x8010:00 ... 0x801F:255
- Channel 2: parameter range 0x8020:00 ... 0x802F:255
- ...

This is generally written as 0x80n0.

Detailed information on the CoE interface can be found in the EtherCAT system documentation on the Beckhoff website.

5 Parameterization and commissioning

5.1 TwinCAT Development Environment

The Software for automation TwinCAT (The Windows Control and Automation Technology) will be distinguished into:

- TwinCAT 2: System Manager (Configuration) & PLC Control (Programming)
- TwinCAT 3: Enhancement of TwinCAT 2 (Programming and Configuration takes place via a common Development Environment)

Details:

- **TwinCAT 2:**
  - Connects I/O devices to tasks in a variable-oriented manner
  - Connects tasks to tasks in a variable-oriented manner
  - Supports units at the bit level
  - Supports synchronous or asynchronous relationships
  - Exchange of consistent data areas and process images
  - Datalink on NT - Programs by open Microsoft Standards (OLE, OCX, ActiveX, DCOM+, etc.)
  - Integration of IEC 61131-3-Software-SPS, Software- NC and Software-CNC within Windows NT/2000/XP/Vista, Windows 7, NT/XP Embedded, CE
  - Interconnection to all common fieldbusses
  - More...

Additional features:

- **TwinCAT 3** (eXtended Automation)**:**
  - Visual-Studio®-Integration
  - Choice of the programming language
  - Supports object orientated extension of IEC 61131-3
  - Usage of C/C++ as programming language for real time applications
  - Connection to MATLAB®/Simulink®
  - Open interface for expandability
  - Flexible run-time environment
  - Active support of Multi-Core- und 64-Bit-Operatingsystem
  - Automatic code generation and project creation with the TwinCAT Automation Interface
  - More...

Within the following sections commissioning of the TwinCAT Development Environment on a PC System for the control and also the basically functions of unique control elements will be explained.

Please see further information to TwinCAT 2 and TwinCAT 3 at http://infosys.beckhoff.com.

5.1.1 Installation of the TwinCAT real-time driver

In order to assign real-time capability to a standard Ethernet port of an IPC controller, the Beckhoff real-time driver has to be installed on this port under Windows.

This can be done in several ways. One option is described here.

In the System Manager call up the TwinCAT overview of the local network interfaces via Options → Show Real Time Ethernet Compatible Devices.

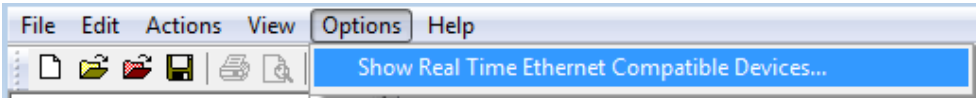


Fig. 34: *System Manager "Options" (TwinCAT 2)*

This have to be called up by the Menü "TwinCAT" within the TwinCAT 3 environment:

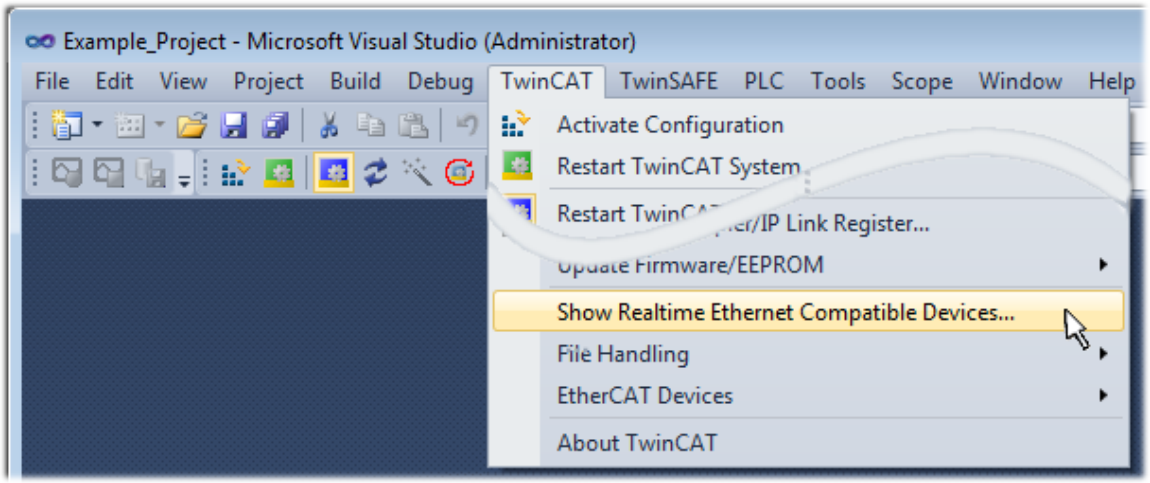


Fig. 35: *Call up under VS Shell (TwinCAT 3)*

The following dialog appears:

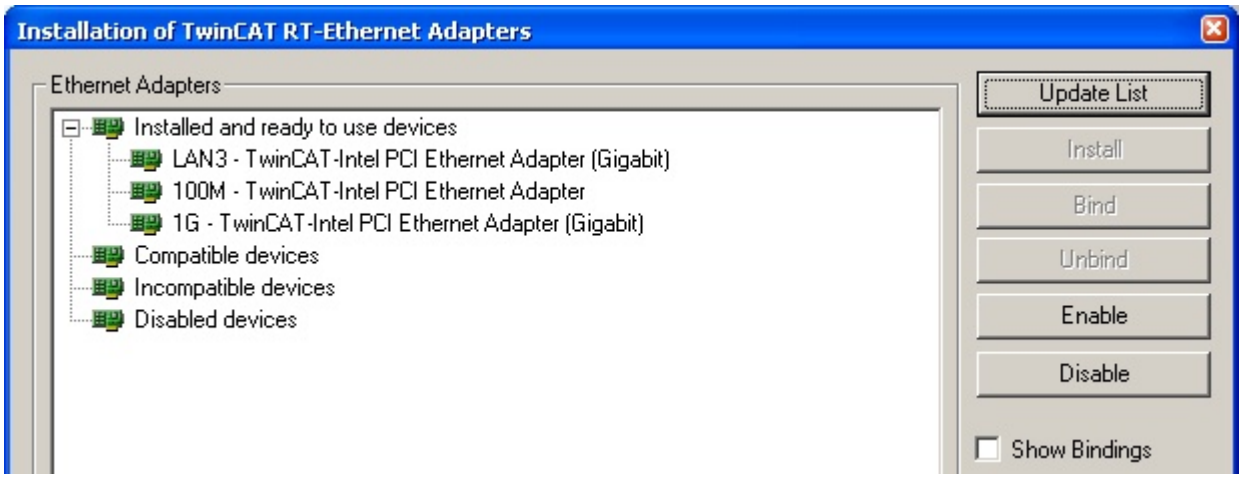


Fig. 36: *Overview of network interfaces*

Interfaces listed under "Compatible devices" can be assigned a driver via the "Install" button. A driver should only be installed on compatible devices.

A Windows warning regarding the unsigned driver can be ignored.

Alternatively an EtherCAT-device can be inserted first of all as described in chapter Offline configuration creation, section "Creating the EtherCAT device" [▶ 52] in order to view the compatible ethernet ports via its EtherCAT properties (tab „Adapter", button „Compatible Devices…"):

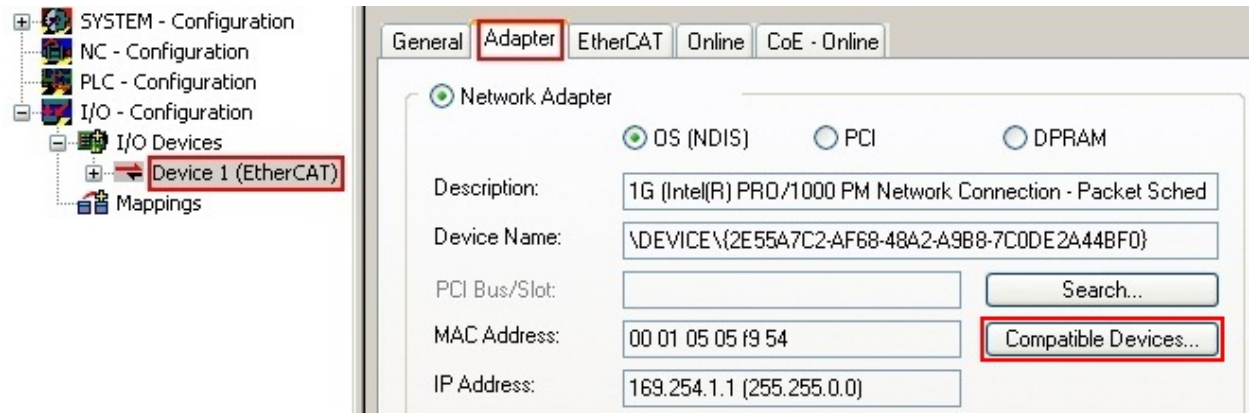
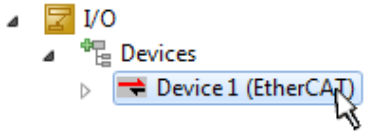


Fig. 37: *EtherCAT device properties(TwinCAT 2): click on „Compatible Devices…“ of tab “Adapter”*

TwinCAT 3: the properties of the EtherCAT device can be opened by double click on “Device .. (EtherCAT)” within the Solution Explorer under “I/O”:



After the installation the driver appears activated in the Windows overview for the network interface (Windows Start → System Properties → Network)

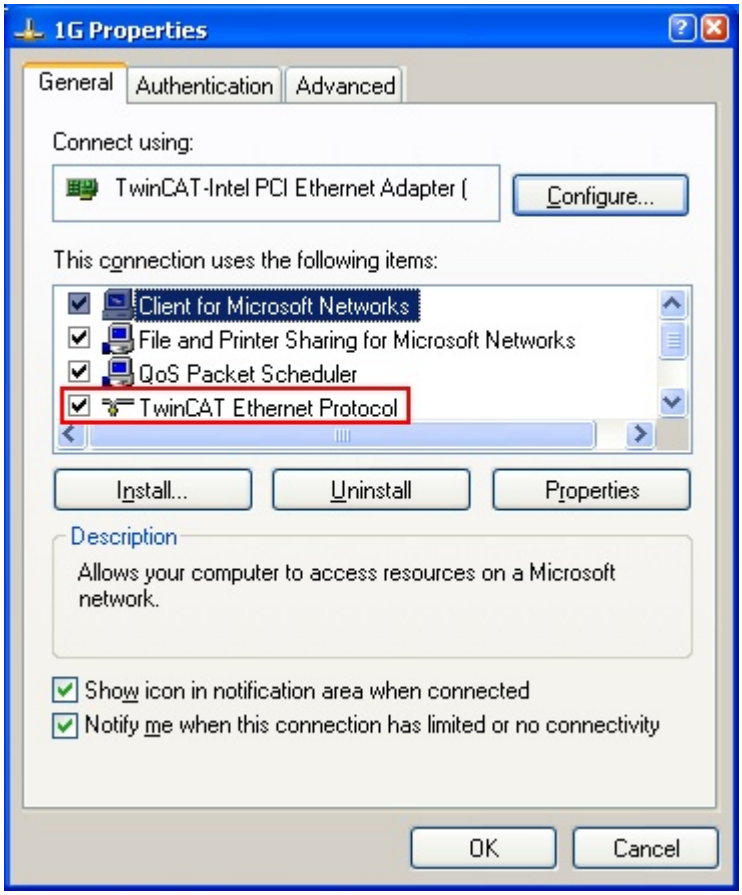


Fig. 38: *Windows properties of the network interface*

A correct setting of the driver could be:

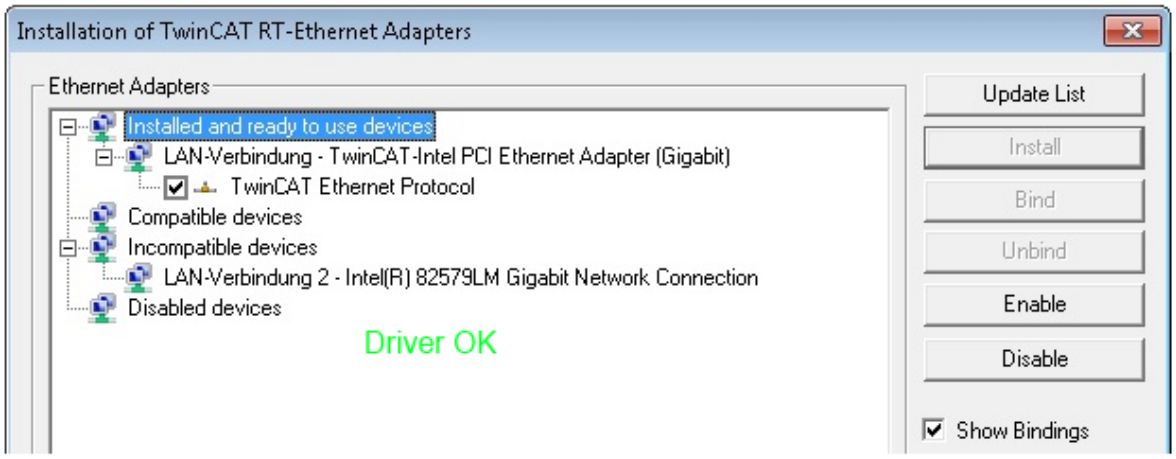


Fig. 39: *Exemplary correct driver setting for the Ethernet port*

Other possible settings have to be avoided:

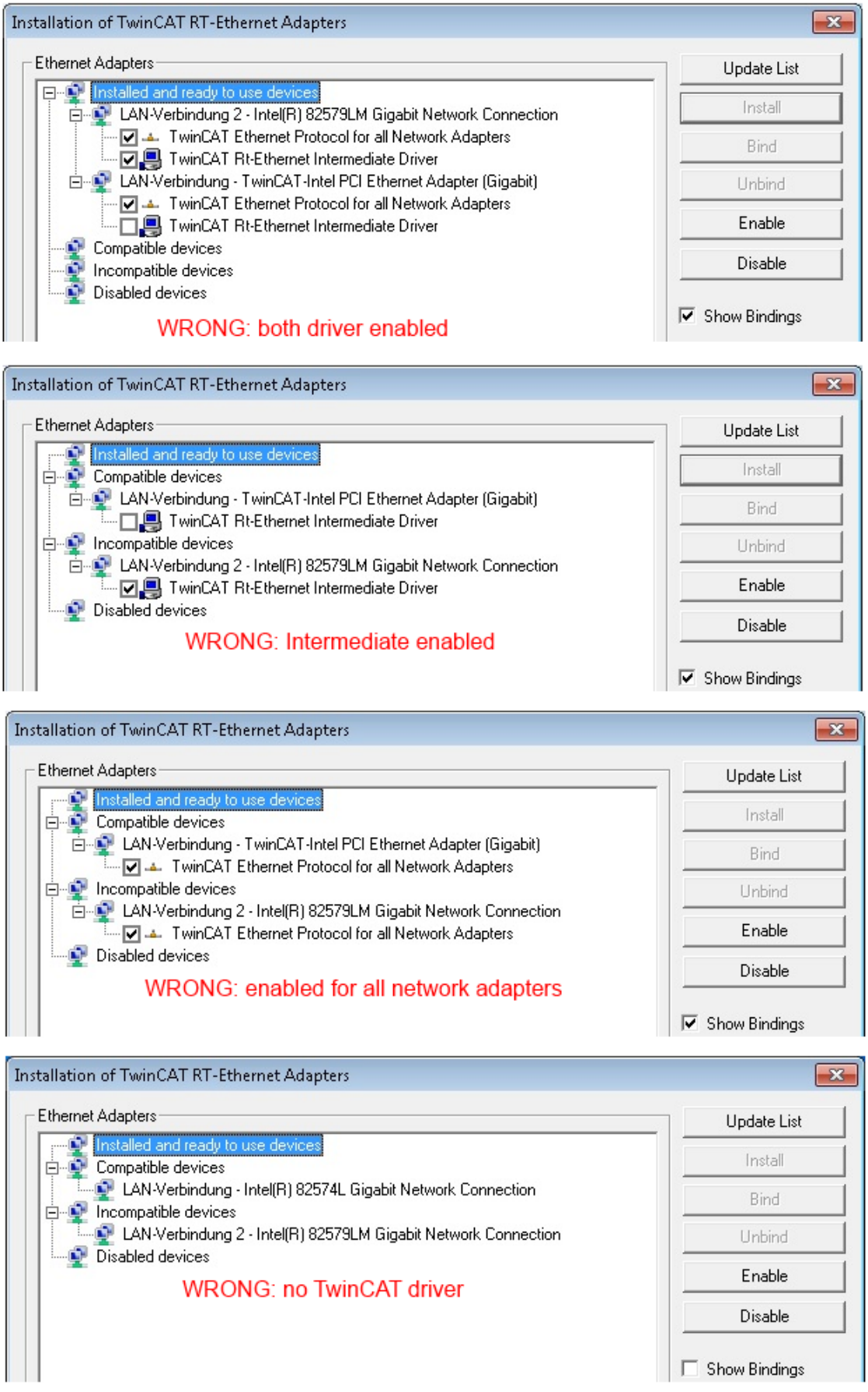


Fig. 40: *Incorrect driver settings for the Ethernet port*

IP address of the port used

IP address/DHCP

In most cases an Ethernet port that is configured as an EtherCAT device will not transport general IP packets. For this reason and in cases where an EL6601 or similar devices are used it is useful to specify a fixed IP address for this port via the “Internet Protocol TCP/IP” driver setting and to disable DHCP. In this way the delay associated with the DHCP client for the Ethernet port assigning itself a default IP address in the absence of a DHCP server is avoided. A suitable address space is 192.168.x.x, for example.

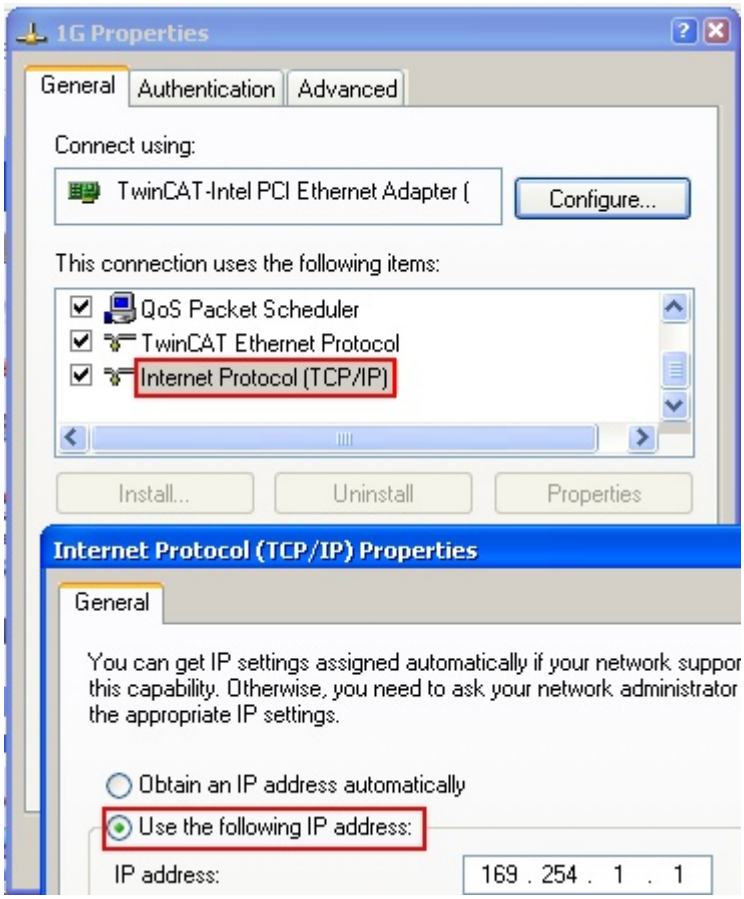


Fig. 41: *TCP/IP setting for the Ethernet port*

5.1.2 Notes regarding ESI device description

Installation of the latest ESI device description

The TwinCAT EtherCAT master/System Manager needs the device description files for the devices to be used in order to generate the configuration in online or offline mode. The device descriptions are contained in the so-called ESI files (EtherCAT Slave Information) in XML format. These files can be requested from the respective manufacturer and are made available for download. An *.xml file may contain several device descriptions.

The ESI files for Beckhoff EtherCAT devices are available on the Beckhoff website.

The ESI files should be stored in the TwinCAT installation directory.

Default settings:

- **TwinCAT 2**: C:\TwinCAT\IO\EtherCAT
- **TwinCAT 3**: C:\TwinCAT\3.1\Config\Io\EtherCAT

The files are read (once) when a new System Manager window is opened, if they have changed since the last time the System Manager window was opened.

A TwinCAT installation includes the set of Beckhoff ESI files that was current at the time when the TwinCAT build was created.

For TwinCAT 2.11/TwinCAT 3 and higher, the ESI directory can be updated from the System Manager, if the programming PC is connected to the Internet; by

- **TwinCAT 2**: Option → “Update EtherCAT Device Descriptions”
- **TwinCAT 3**: TwinCAT → EtherCAT Devices → “Update Device Descriptions (via ETG Website)…”

The TwinCAT ESI Updater is available for this purpose.

ESI

The *.xml files are associated with *.xsd files, which describe the structure of the ESI XML files. To update the ESI device descriptions, both file types should therefore be updated.

Device differentiation

EtherCAT devices/slaves are distinguished by four properties, which determine the full device identifier. For example, the device identifier EL2521-0025-1018 consists of:

- family key “EL”
- name “2521”
- type “0025”
- and revision “1018”

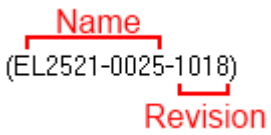


Fig. 42: *Identifier structure*

The order identifier consisting of name + type (here: EL2521-0010) describes the device function. The revision indicates the technical progress and is managed by Beckhoff. In principle, a device with a higher revision can replace a device with a lower revision, unless specified otherwise, e.g. in the documentation. Each revision has its own ESI description. See further notes [▶ 7].

Online description

If the EtherCAT configuration is created online through scanning of real devices (see section Online setup) and no ESI descriptions are available for a slave (specified by name and revision) that was found, the System Manager asks whether the description stored in the device should be used. In any case, the System Manager needs this information for setting up the cyclic and acyclic communication with the slave correctly.

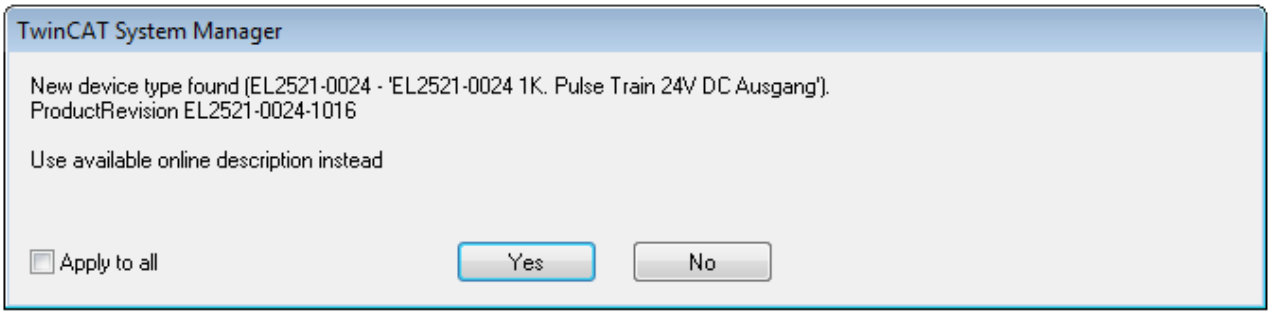


Fig. 43: *OnlineDescription information window (TwinCAT 2)*

In TwinCAT 3 a similar window appears, which also offers the Web update:

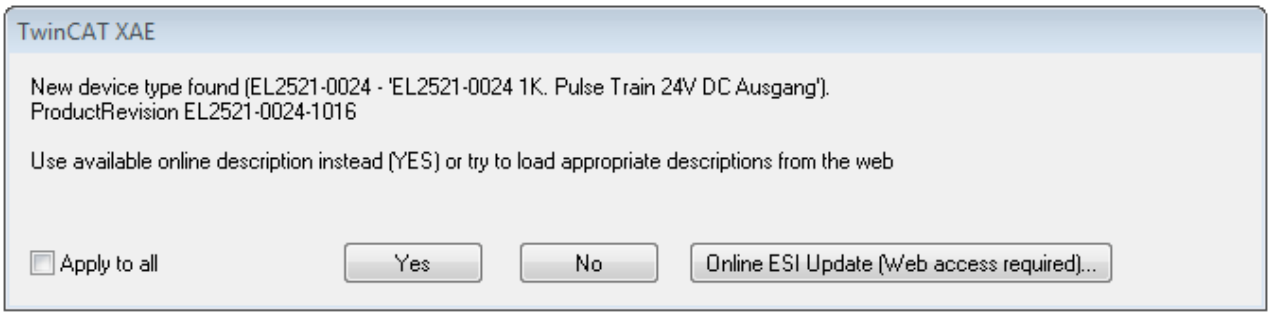


Fig. 44: *Information window OnlineDescription (TwinCAT 3)*

If possible, the *Yes* is to be rejected and the required ESI is to be requested from the device manufacturer. After installation of the XML/XSD file the configuration process should be repeated.

NOTE

Changing the 'usual' configuration through a scan

✓ If a scan discovers a device that is not yet known to TwinCAT, distinction has to be made between two cases. Taking the example here of the EL2521-0000 in the revision 1019

a) no ESI is present for the EL2521-0000 device at all, either for the revision 1019 or for an older revision. The ESI must then be requested from the manufacturer (in this case Beckhoff).

b) an ESI is present for the EL2521-0000 device, but only in an older revision, e.g. 1018 or 1017.
In this case an in-house check should first be performed to determine whether the spare parts stock allows the integration of the increased revision into the configuration at all. A new/higher revision usually also brings along new features. If these are not to be used, work can continue without reservations with the previous revision 1018 in the configuration. This is also stated by the Beckhoff compatibility rule.

Refer in particular to the chapter 'General notes on the use of Beckhoff EtherCAT IO components' and for manual configuration to the chapter 'Offline configuration creation' [▶ 52].

If the OnlineDescription is used regardless, the System Manager reads a copy of the device description from the EEPROM in the EtherCAT slave. In complex slaves the size of the EEPROM may not be sufficient for the complete ESI, in which case the ESI would be *incomplete* in the configurator. Therefore it's recommended using an offline ESI file with priority in such a case.

The System Manager creates for online recorded device descriptions a new file "OnlineDescription0000...xml" in its ESI directory, which contains all ESI descriptions that were read online.

OnlineDescriptionCache00000002.xml

Fig. 45: *File OnlineDescription.xml created by the System Manager*

Is a slave desired to be added manually to the configuration at a later stage, online created slaves are indicated by a prepended symbol “>” in the selection list (see Figure *“Indication of an online recorded ESI of EL2521 as an example”*).

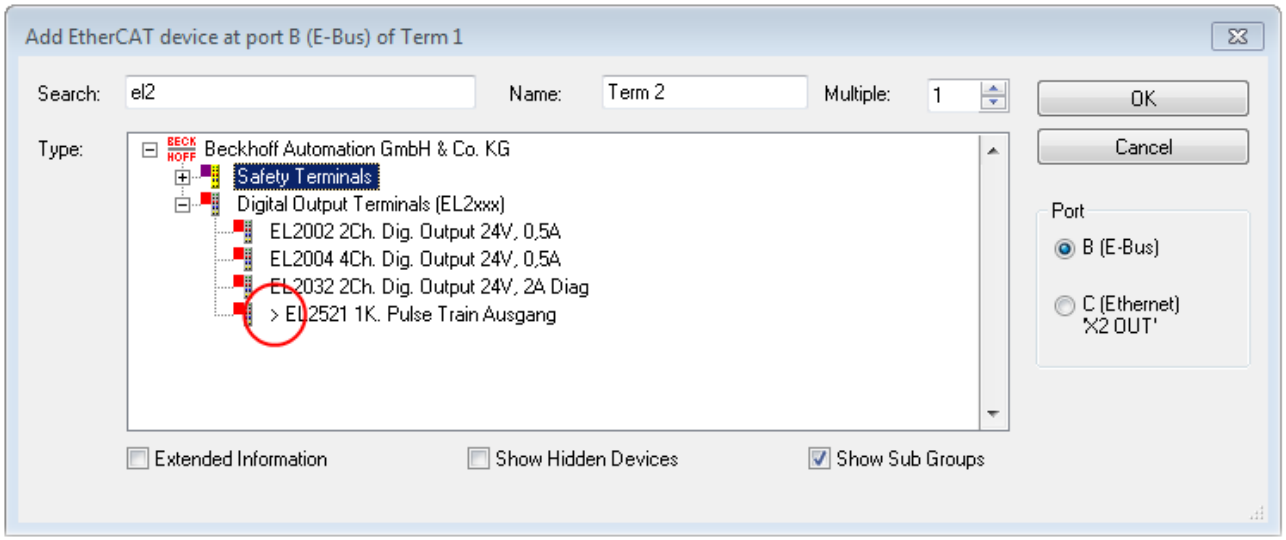


Fig. 46: *Indication of an online recorded ESI of EL2521 as an example*

If such ESI files are used and the manufacturer's files become available later, the file OnlineDescription.xml should be deleted as follows:

- close all System Manager windows
- restart TwinCAT in Config mode
- delete "OnlineDescription0000...xml"
- restart TwinCAT System Manager

This file should not be visible after this procedure, if necessary press <F5> to update

OnlineDescription for TwinCAT 3.x

In addition to the file described above "OnlineDescription0000...xml" , a so called EtherCAT cache with new discovered devices is created by TwinCAT 3.x, e.g. under Windows 7:

C:\User\[USERNAME]\AppData\Roaming\Beckhoff\TwinCAT3\Components\Base\EtherCATCache.xml

(Please note the language settings of the OS!)
You have to delete this file, too.

Faulty ESI file

If an ESI file is faulty and the System Manager is unable to read it, the System Manager brings up an information window.

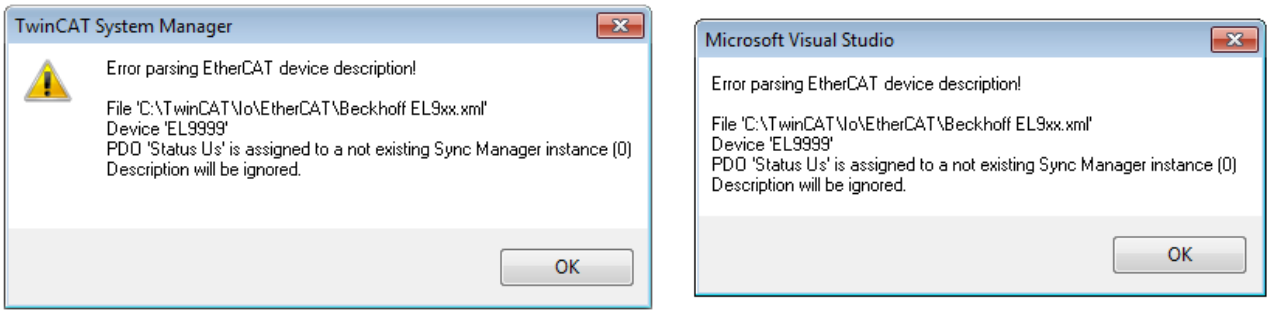


Fig. 47: *Information window for faulty ESI file (left: TwinCAT 2; right: TwinCAT 3)*

Reasons may include:

- Structure of the *.xml does not correspond to the associated *.xsd file → check your schematics
- Contents cannot be translated into a device description → contact the file manufacturer

5.1.3 OFFLINE configuration creation

Creating the EtherCAT device

Create an EtherCAT device in an empty System Manager window.

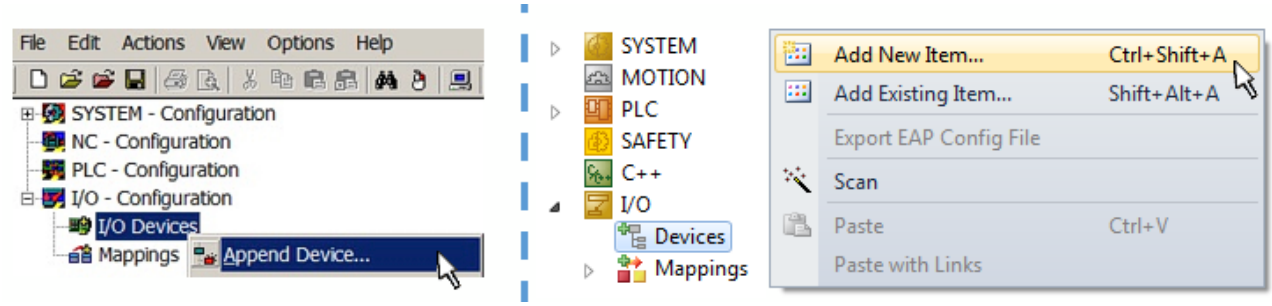


Fig. 48: *Append EtherCAT device (left: TwinCAT 2; right: TwinCAT 3)*

Select type ‘EtherCAT’ for an EtherCAT I/O application with EtherCAT slaves. For the present publisher/subscriber service in combination with an EL6601/EL6614 terminal select “EtherCAT Automation Protocol via EL6601”.

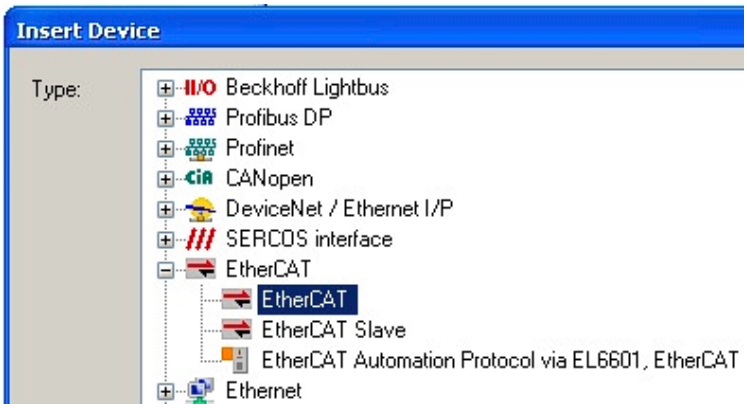


Fig. 49: *Selecting the EtherCAT connection (TwinCAT 2.11, TwinCAT 3)*

Then assign a real Ethernet port to this virtual device in the runtime system.

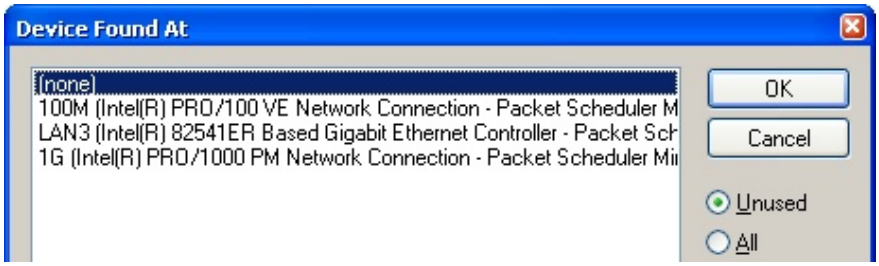


Fig. 50: *Selecting the Ethernet port*

This query may appear automatically when the EtherCAT device is created, or the assignment can be set/modified later in the properties dialog; see Fig. *“EtherCAT device properties (TwinCAT 2)”*.

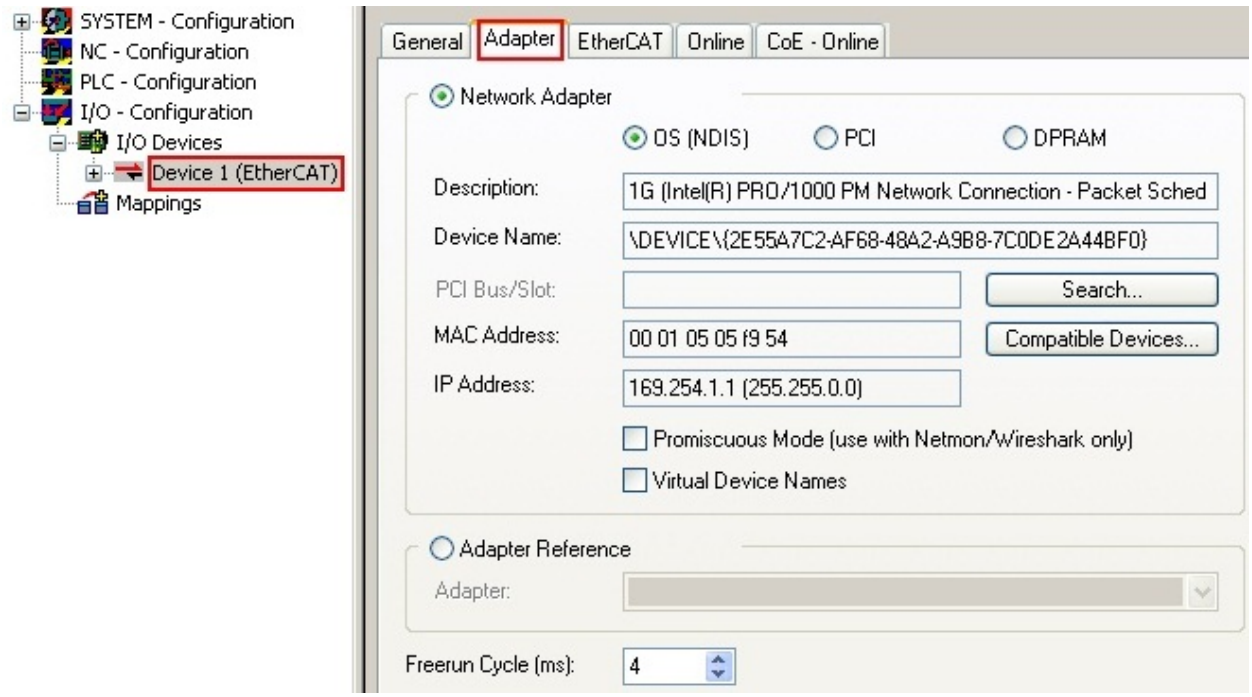
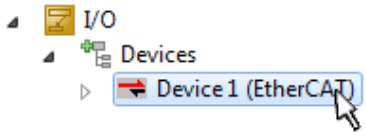


Fig. 51: *EtherCAT device properties (TwinCAT 2)*

TwinCAT 3: the properties of the EtherCAT device can be opened by double click on “Device .. (EtherCAT)” within the Solution Explorer under “I/O”:



Selecting the Ethernet port

Ethernet ports can only be selected for EtherCAT devices for which the TwinCAT real-time driver is installed. This has to be done separately for each port. Please refer to the respective installation page [▶ 42].

Defining EtherCAT slaves

Further devices can be appended by right-clicking on a device in the configuration tree.

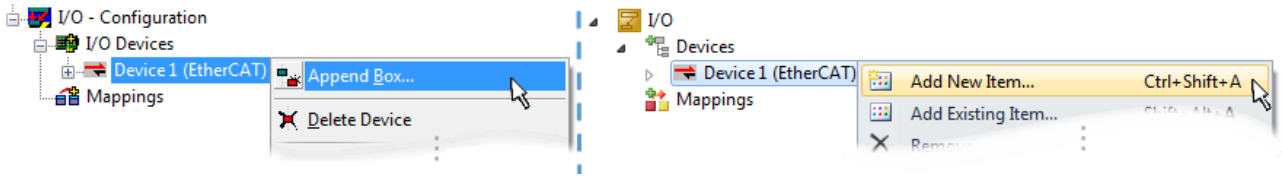


Fig. 52: *Appending EtherCAT devices (left: TwinCAT 2; right: TwinCAT 3)*

The dialog for selecting a new device opens. Only devices for which ESI files are available are displayed.

Only devices are offered for selection that can be appended to the previously selected device. Therefore the physical layer available for this port is also displayed (Fig. *“Selection dialog for new EtherCAT device”*, A). In the case of cable-based Fast-Ethernet physical layer with PHY transfer, then also only cable-based devices are available, as shown in Fig. *“Selection dialog for new EtherCAT device”*. If the preceding device has several free ports (e.g. EK1122 or EK1100), the required port can be selected on the right-hand side (A).

Overview of physical layer

- “Ethernet”: cable-based 100BASE-TX: EK couplers, EP boxes, devices with RJ45/M8/M12 connector
- “E-Bus”: LVDS “terminal bus”, “EJ-module”: EL/ES terminals, various modular modules

The search field facilitates finding specific devices (since TwinCAT 2.11 or TwinCAT 3).

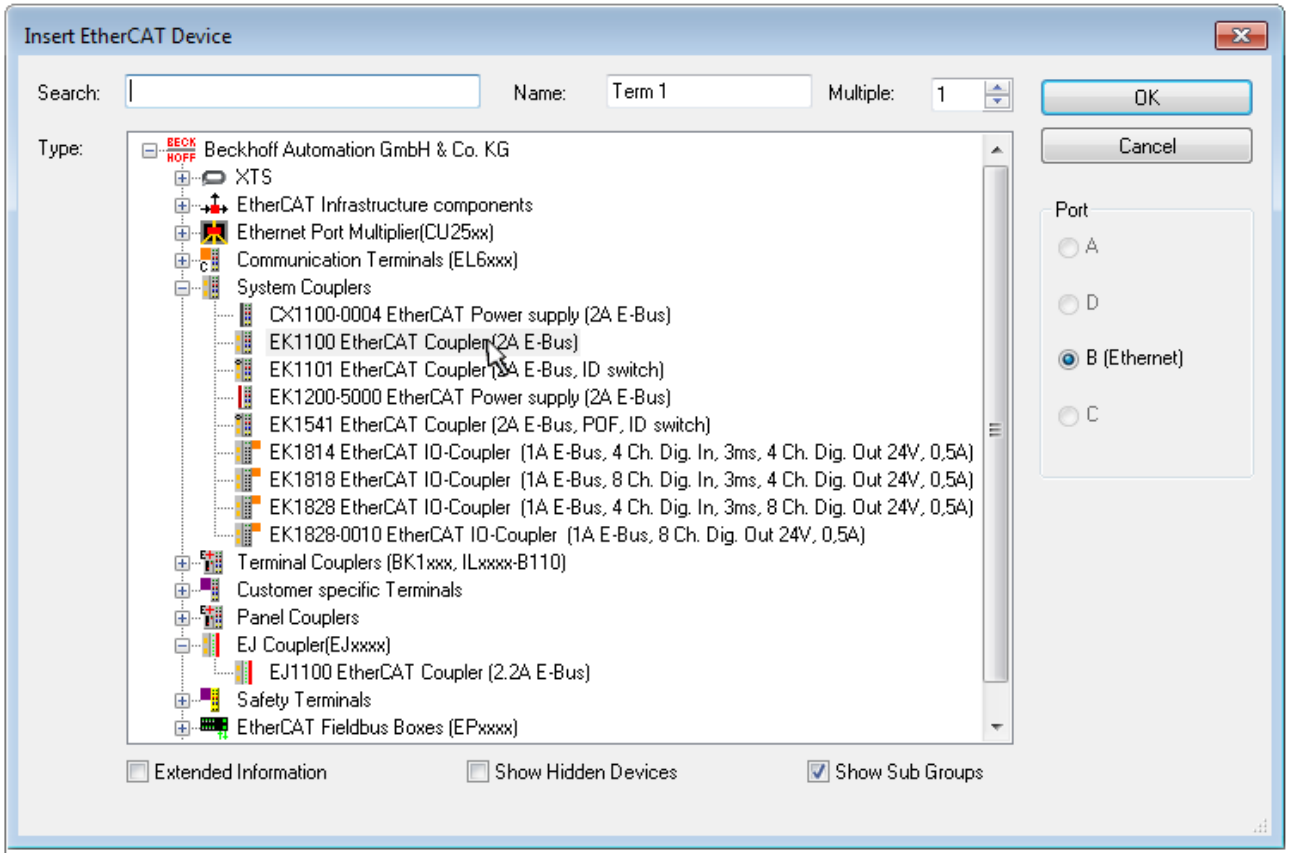


Fig. 53: *Selection dialog for new EtherCAT device*

By default only the name/device type is used as selection criterion. For selecting a specific revision of the device the revision can be displayed as “Extended Information”.

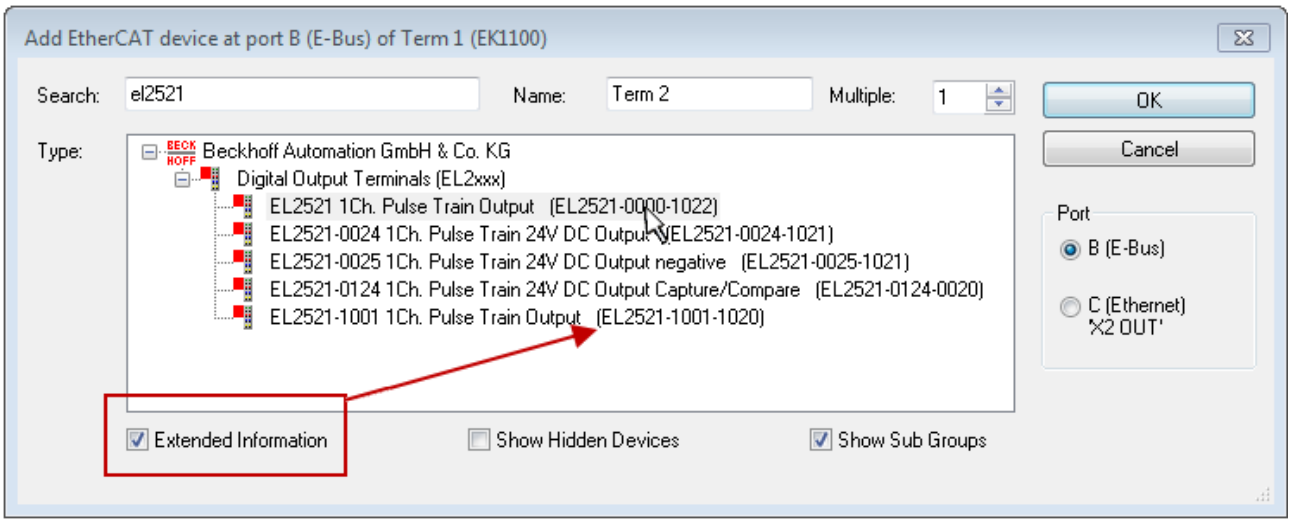


Fig. 54: *Display of device revision*

In many cases several device revisions were created for historic or functional reasons, e.g. through technological advancement. For simplification purposes (see Fig. *“Selection dialog for new EtherCAT device”*) only the last (i.e. highest) revision and therefore the latest state of production is displayed in the selection dialog for Beckhoff devices. To show all device revisions available in the system as ESI descriptions tick the “Show Hidden Devices” check box, see Fig. *“Display of previous revisions”*.

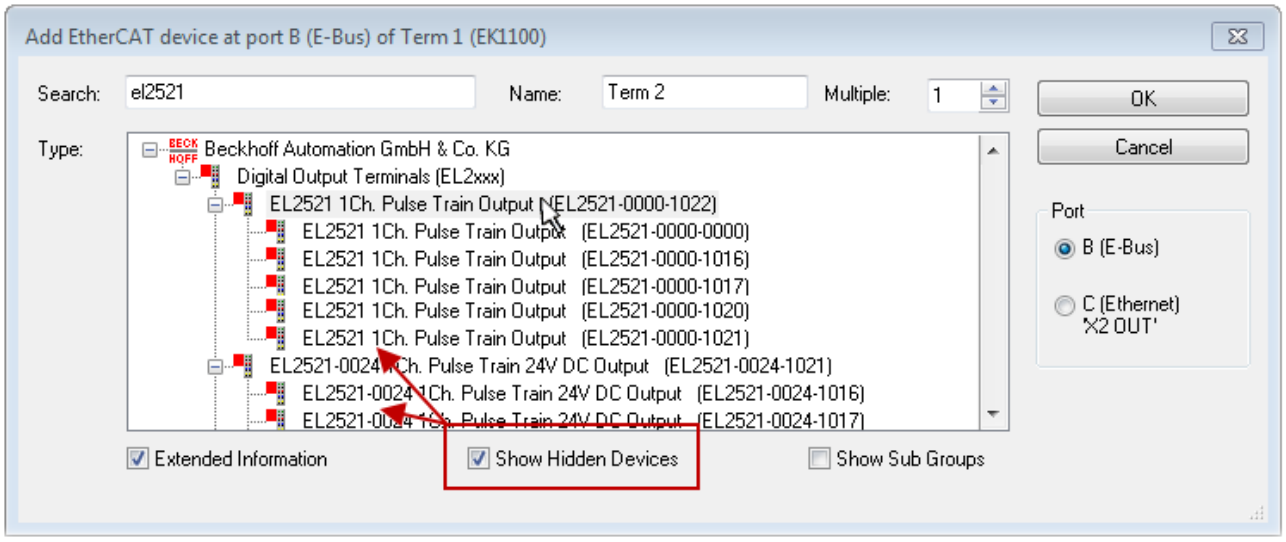


Fig. 55: *Display of previous revisions*

Device selection based on revision, compatibility

The ESI description also defines the process image, the communication type between master and slave/device and the device functions, if applicable. The physical device (firmware, if available) has to support the communication queries/settings of the master. This is backward compatible, i.e. newer devices (higher revision) should be supported if the EtherCAT master addresses them as an older revision. The following compatibility rule of thumb is to be assumed for Beckhoff EtherCAT Terminals/ Boxes/ EJ-modules:

device revision in the system >= device revision in the configuration

This also enables subsequent replacement of devices without changing the configuration (different specifications are possible for drives).

Example:

If an EL2521-0025-**1018** is specified in the configuration, an EL2521-0025-**1018** or higher (-**1019**, -**1020**) can be used in practice.

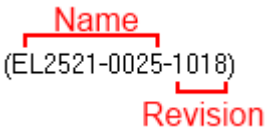


Fig. 56: *Name/revision of the terminal*

If current ESI descriptions are available in the TwinCAT system, the last revision offered in the selection dialog matches the Beckhoff state of production. It is recommended to use the last device revision when creating a new configuration, if current Beckhoff devices are used in the real application. Older revisions should only be used if older devices from stock are to be used in the application.

In this case the process image of the device is shown in the configuration tree and can be parameterised as follows: linking with the task, CoE/DC settings, plug-in definition, startup settings, ...

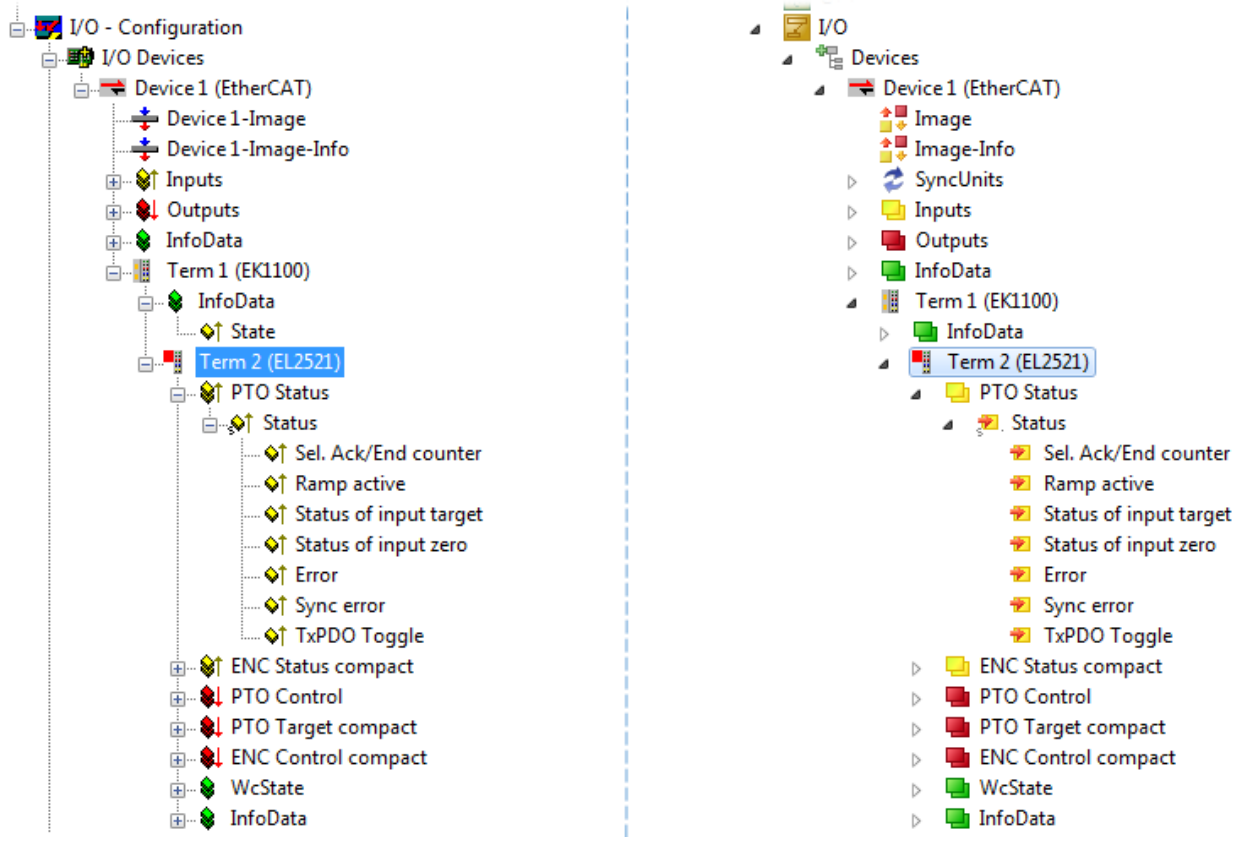


Fig. 57: *EtherCAT terminal in the TwinCAT tree (left: TwinCAT 2; right: TwinCAT 3)*

5.1.4 ONLINE configuration creation

Detecting/scanning of the EtherCAT device

The online device search can be used if the TwinCAT system is in CONFIG mode. This can be indicated by a symbol right below in the information bar:

- on TwinCAT 2 by a blue display “Config Mode” within the System Manager window: Config Mode .
- on TwinCAT 3 within the user interface of the development environment by a symbol .

TwinCAT can be set into this mode:

- TwinCAT 2: by selection of in the Menubar or by “Actions” → “Set/Reset TwinCAT to Config Mode…”
- TwinCAT 3: by selection of in the Menubar or by „TwinCAT“ → “Restart TwinCAT (Config Mode)“

Online scanning in Config mode

The online search is not available in RUN mode (production operation). Note the differentiation between TwinCAT programming system and TwinCAT target system.

The TwinCAT 2 icon ( ) or TwinCAT 3 icon ( ) within the Windows-Taskbar always shows the TwinCAT mode of the local IPC. Compared to that, the System Manager window of TwinCAT 2 or the user interface of TwinCAT 3 indicates the state of the target system.

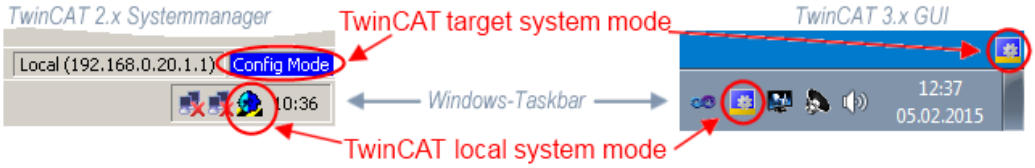


Fig. 58: *Differentiation local/target system (left: TwinCAT 2; right: TwinCAT 3)*

Right-clicking on “I/O Devices” in the configuration tree opens the search dialog.

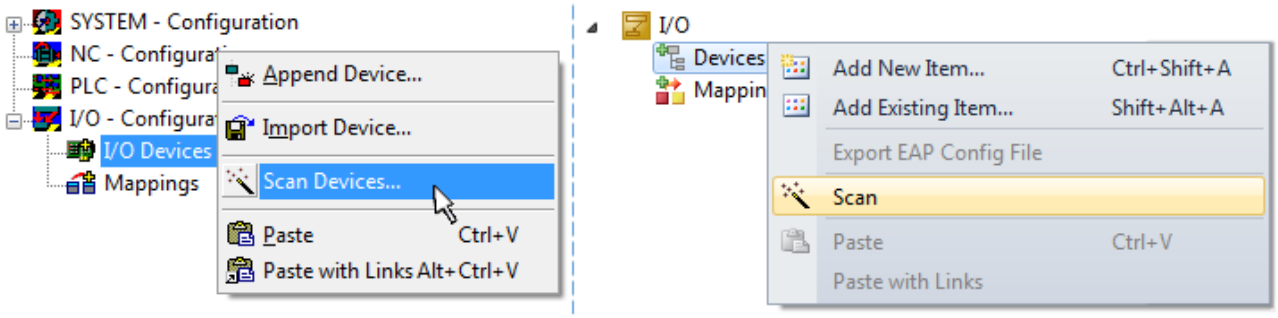


Fig. 59: *Scan Devices (left: TwinCAT 2; right: TwinCAT 3)*

This scan mode attempts to find not only EtherCAT devices (or Ethernet ports that are usable as such), but also NOVRAM, fieldbus cards, SMB etc. However, not all devices can be found automatically.

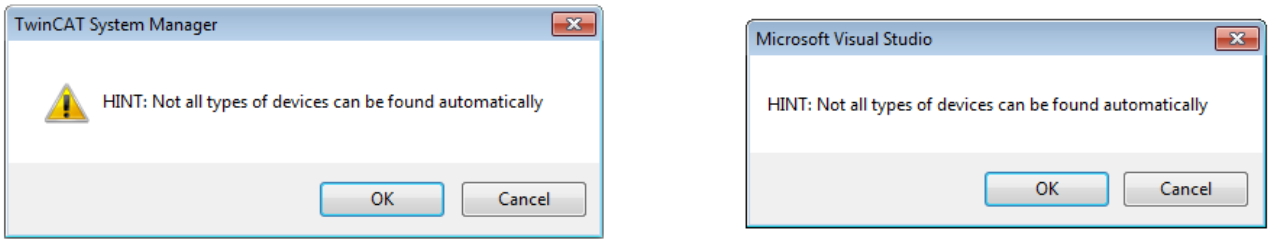


Fig. 60: *Note for automatic device scan (left: TwinCAT 2; right: TwinCAT 3)*

Ethernet ports with installed TwinCAT real-time driver are shown as “RT Ethernet” devices. An EtherCAT frame is sent to these ports for testing purposes. If the scan agent detects from the response that an EtherCAT slave is connected, the port is immediately shown as an “EtherCAT Device” .

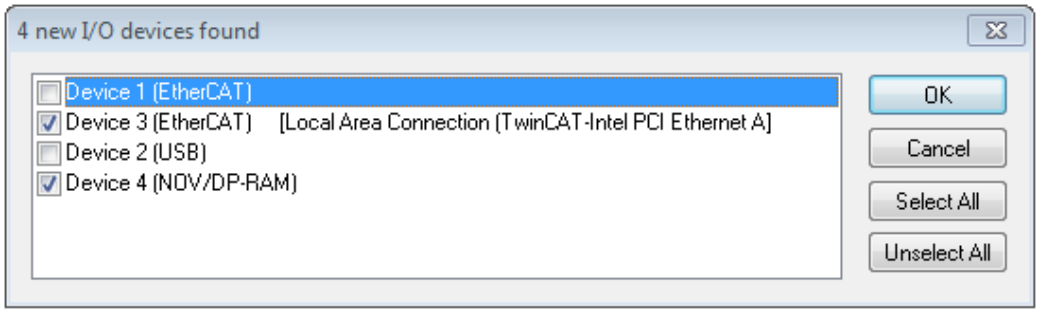


Fig. 61: *Detected Ethernet devices*

Via respective checkboxes devices can be selected (as illustrated in Fig. “*Detected Ethernet devices*” e.g. Device 3 and Device 4 were chosen). After confirmation with “OK” a device scan is suggested for all selected devices, see Fig.: “*Scan query after automatic creation of an EtherCAT device”*.

Selecting the Ethernet port

Ethernet ports can only be selected for EtherCAT devices for which the TwinCAT real-time driver is installed. This has to be done separately for each port. Please refer to the respective installation page [▶ 42].

Detecting/Scanning the EtherCAT devices

Online scan functionality

During a scan the master queries the identity information of the EtherCAT slaves from the slave EEPROM. The name and revision are used for determining the type. The respective devices are located in the stored ESI data and integrated in the configuration tree in the default state defined there.

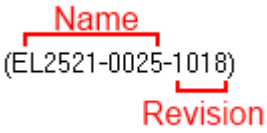


Fig. 62: *Example default state*

NOTE

Slave scanning in practice in series machine production

The scanning function should be used with care. It is a practical and fast tool for creating an initial configuration as a basis for commissioning. In series machine production or reproduction of the plant, however, the function should no longer be used for the creation of the configuration, but if necessary for comparison [▶ 62] with the defined initial configuration.Background: since Beckhoff occasionally increases the revision version of the delivered products for product maintenance reasons, a configuration can be created by such a scan which (with an identical machine construction) is identical according to the device list; however, the respective device revision may differ from the initial configuration.

Example:

Company A builds the prototype of a machine B, which is to be produced in series later on. To do this the prototype is built, a scan of the IO devices is performed in TwinCAT and the initial configuration ‘B.tsm’ is created. The EL2521-0025 EtherCAT terminal with the revision 1018 is located somewhere. It is thus built into the TwinCAT configuration in this way:

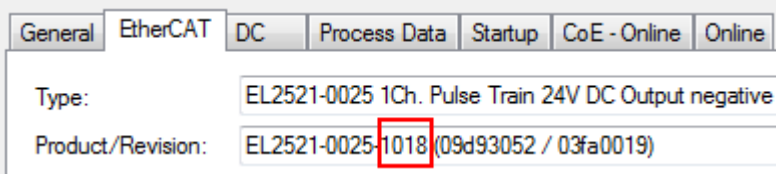


Fig. 63: *Installing EthetCAT terminal with revision -1018*

Likewise, during the prototype test phase, the functions and properties of this terminal are tested by the programmers/commissioning engineers and used if necessary, i.e. addressed from the PLC ‘B.pro’ or the NC. (the same applies correspondingly to the TwinCAT 3 solution files).

The prototype development is now completed and series production of machine B starts, for which Beckhoff continues to supply the EL2521-0025-0018. If the commissioning engineers of the series machine production department always carry out a scan, a B configuration with the identical contents results again for each machine. Likewise, A might create spare parts stores worldwide for the coming series-produced machines with EL2521-0025-1018 terminals.

After some time Beckhoff extends the EL2521-0025 by a new feature C. Therefore the FW is changed, outwardly recognizable by a higher FW version and **a new revision -1019**. Nevertheless the new device naturally supports functions and interfaces of the predecessor version(s); an adaptation of ‘B.tsm’ or even ‘B.pro’ is therefore unnecessary. The series-produced machines can continue to be built with ‘B.tsm’ and ‘B.pro’; it makes sense to perform a comparative scan [▶ 62] against the initial configuration ‘B.tsm’ in order to check the built machine.

However, if the series machine production department now doesn’t use ‘B.tsm’, but instead carries out a scan to create the productive configuration, the revision **-1019** is automatically detected and built into the configuration:

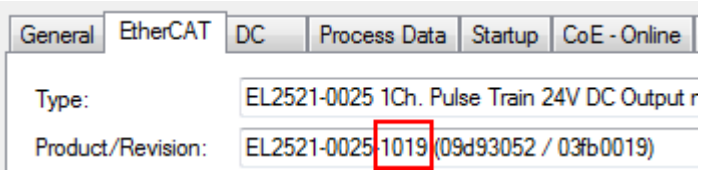


Fig. 64: *Detection of EtherCAT terminal with revision -1019*

This is usually not noticed by the commissioning engineers. TwinCAT cannot signal anything either, since virtually a new configuration is created. According to the compatibility rule, however, this means that no EL2521-0025-**1018** should be built into this machine as a spare part (even if this nevertheless works in the vast majority of cases).

In addition, it could be the case that, due to the development accompanying production in company A, the new feature C of the EL2521-0025-1019 (for example, an improved analog filter or an additional process data for the diagnosis) is discovered and used without in-house consultation. The previous stock of spare part devices are then no longer to be used for the new configuration ‘B2.tsm’ created in this way.Þ if series machine production is established, the scan should only be performed for informative purposes for comparison with a defined initial configuration. Changes are to be made with care!

If an EtherCAT device was created in the configuration (manually or through a scan), the I/O field can be scanned for devices/slaves.

Fig. 65: *Scan query after automatic creation of an EtherCAT device (left: TwinCAT 2; right: TwinCAT 3)*

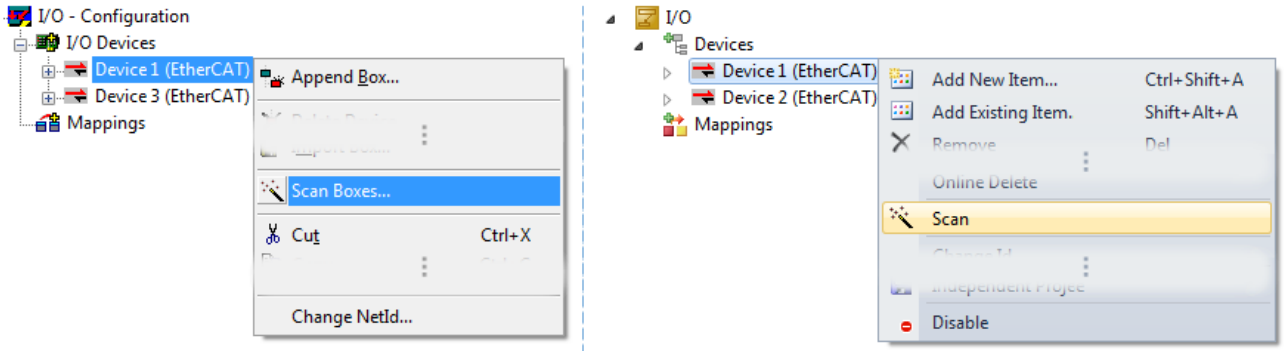


Fig. 66: *Manual triggering of a device scan on a specified EtherCAT device (left: TwinCAT 2; right: TwinCAT 3)*

In the System Manager (TwinCAT 2) or the User Interface (TwinCAT 3) the scan process can be monitored via the progress bar at the bottom in the status bar.

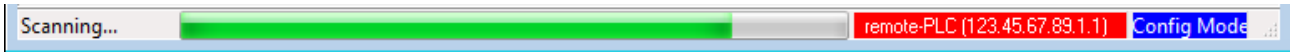


Fig. 67: *Scan progressexemplary by TwinCAT 2*

The configuration is established and can then be switched to online state (OPERATIONAL).

Fig. 68: *Config/FreeRun query (left: TwinCAT 2; right: TwinCAT 3)*

In Config/FreeRun mode the System Manager display alternates between blue and red, and the EtherCAT device continues to operate with the idling cycle time of 4 ms (default setting), even without active task (NC, PLC).

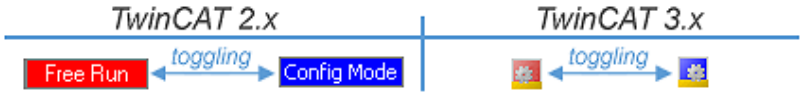


Fig. 69: *Displaying of “Free Run” and “Config Mode” toggling right below in the status bar*

Fig. 70: *TwinCAT can also be switched to this state by using a button (left: TwinCAT 2; right: TwinCAT 3)*

The EtherCAT system should then be in a functional cyclic state, as shown in Fig. *“Online display example”*.

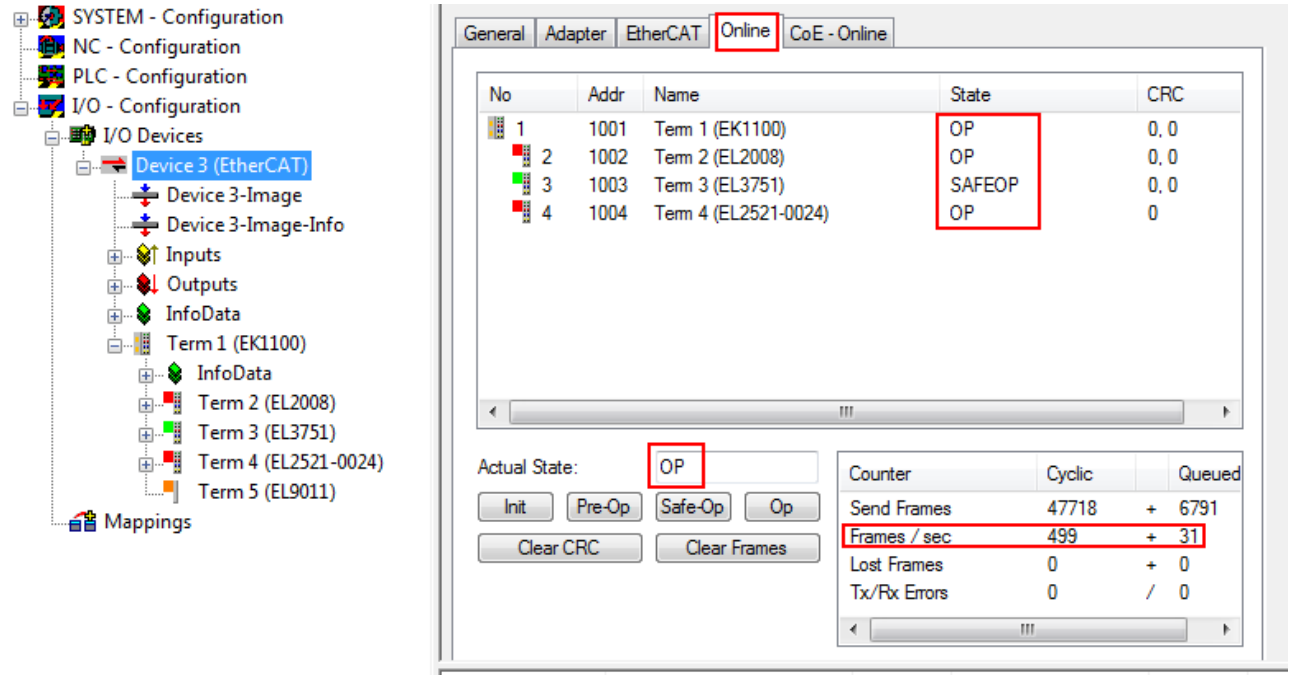


Fig. 71: *Online display example*

Please note:

- all slaves should be in OP state
- the EtherCAT master should be in “Actual State” OP
- “frames/sec” should match the cycle time taking into account the sent number of frames
- no excessive “LostFrames” or CRC errors should occur

The configuration is now complete. It can be modified as described under manual procedure [▶ 52].

Troubleshooting

Various effects may occur during scanning.

- An **unknown device** is detected, i.e. an EtherCAT slave for which no ESI XML description is available. In this case the System Manager offers to read any ESI that may be stored in the device. This case is described in the chapter "Notes regarding ESI device description".
- **Device are not detected properly**
  Possible reasons include:
  - faulty data links, resulting in data loss during the scan
  - slave has invalid device description
  The connections and devices should be checked in a targeted manner, e.g. via the emergency scan. Then re-run the scan.

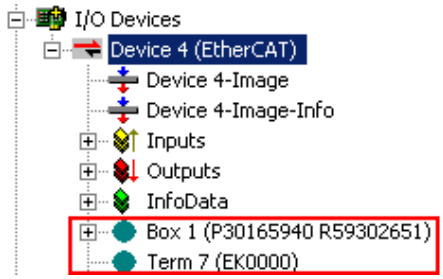


Fig. 72: *Faulty identification*

In the System Manager such devices may be set up as EK0000 or unknown devices. Operation is not possible or meaningful.

Scan over existing Configuration

NOTE

Change of the configuration after comparison

With this scan (TwinCAT 2.11 or 3.1) only the device properties vendor (manufacturer), device name and revision are compared at present! A ‘ChangeTo’ or ‘Copy’ should only be carried out with care, taking into consideration the Beckhoff IO compatibility rule (see above). The device configuration is then replaced by the revision found; this can affect the supported process data and functions.

If a scan is initiated for an existing configuration, the actual I/O environment may match the configuration exactly or it may differ. This enables the configuration to be compared.

Fig. 73: *Identical configuration (left: TwinCAT 2; right: TwinCAT 3)*

If differences are detected, they are shown in the correction dialog, so that the user can modify the configuration as required.

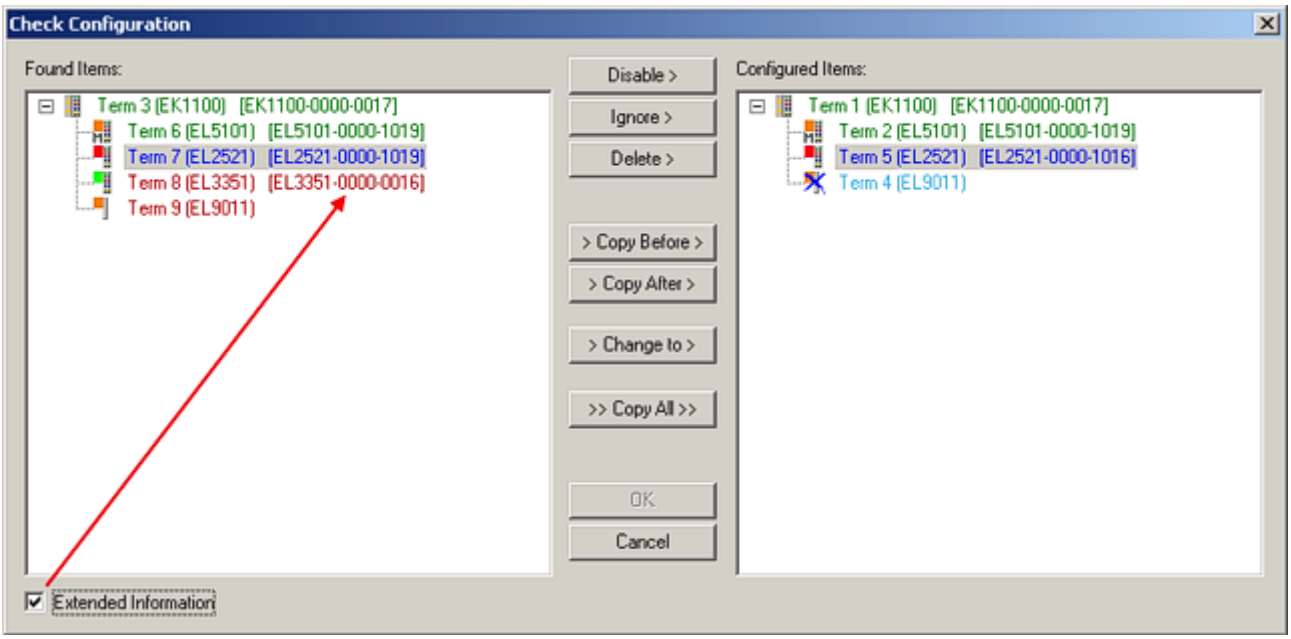


Fig. 74: *Correction dialog*

It is advisable to tick the “Extended Information” check box to reveal differences in the revision.

Colour	Explanation
green	This EtherCAT slave matches the entry on the other side. Both type and revision match.
blue	This EtherCAT slave is present on the other side, but in a different revision. This other revision can have other default values for the process data as well as other/additional functions. If the found revision is higher than the configured revision, the slave may be used provided compatibility issues are taken into account. If the found revision is lower than the configured revision, it is likely that the slave cannot be used. The found device may not support all functions that the master expects based on the higher revision number.
light blue	This EtherCAT slave is ignored (“Ignore” button)
red	• This EtherCAT slave is not present on the other side. • It is present, but in a different revision, which also differs in its properties from the one specified. The compatibility principle then also applies here: if the found revision is higher than the configured revision, use is possible provided compatibility issues are taken into account, since the successor devices should support the functions of the predecessor devices. If the found revision is lower than the configured revision, it is likely that the slave cannot be used. The found device may not support all functions that the master expects based on the higher revision number.

Device selection based on revision, compatibility

The ESI description also defines the process image, the communication type between master and slave/device and the device functions, if applicable. The physical device (firmware, if available) has to support the communication queries/settings of the master. This is backward compatible, i.e. newer devices (higher revision) should be supported if the EtherCAT master addresses them as an older revision. The following compatibility rule of thumb is to be assumed for Beckhoff EtherCAT Terminals/ Boxes/ EJ-modules:

device revision in the system >= device revision in the configuration

This also enables subsequent replacement of devices without changing the configuration (different specifications are possible for drives).

Example:

If an EL2521-0025-**1018** is specified in the configuration, an EL2521-0025-**1018** or higher (-**1019**, -**1020**) can be used in practice.

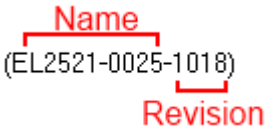


Fig. 75: *Name/revision of the terminal*

If current ESI descriptions are available in the TwinCAT system, the last revision offered in the selection dialog matches the Beckhoff state of production. It is recommended to use the last device revision when creating a new configuration, if current Beckhoff devices are used in the real application. Older revisions should only be used if older devices from stock are to be used in the application.

In this case the process image of the device is shown in the configuration tree and can be parameterised as follows: linking with the task, CoE/DC settings, plug-in definition, startup settings, ...

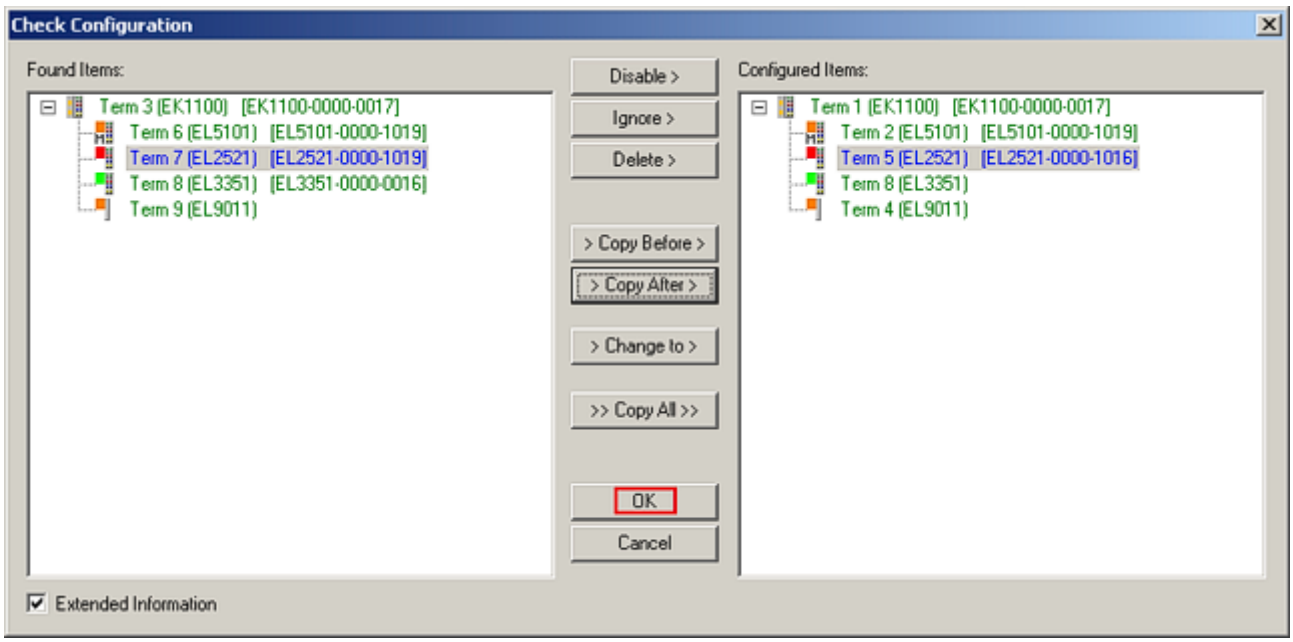


Fig. 76: *Correction dialog with modifications*

Once all modifications have been saved or accepted, click “OK” to transfer them to the real *.tsm configuration.

Change to Compatible Type

TwinCAT offers a function *“Change to Compatible Type…”* for the exchange of a device whilst retaining the links in the task.

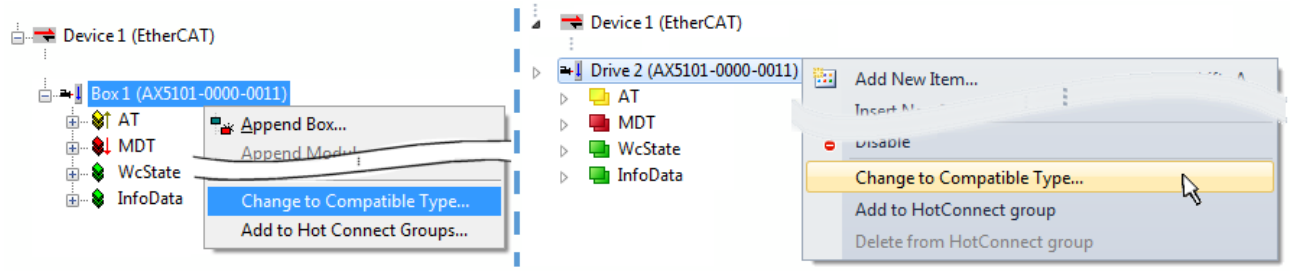


Fig. 77: *Dialog “Change to Compatible Type…” (left: TwinCAT 2; right: TwinCAT 3)*

This function is preferably to be used on AX5000 devices.

Change to Alternative Type

The TwinCAT System Manager offers a function for the exchange of a device: *Change to Alternative Type*

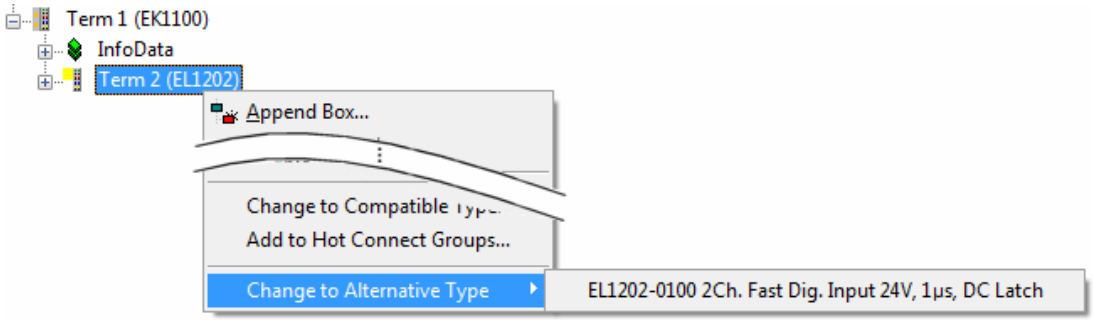


Fig. 78: *TwinCAT 2 Dialog Change to Alternative Type*

If called, the System Manager searches in the procured device ESI (in this example: EL1202-0000) for details of compatible devices contained there. The configuration is changed and the ESI-EEPROM is overwritten at the same time – therefore this process is possible only in the online state (ConfigMode).

5.1.5 EtherCAT slave process data settings

The process data transferred by an EtherCAT slave during each cycle (**P**rocess **D**ata **O**bjects, PDOs) are user data which the application expects to be updated cyclically or which are sent to the slave. To this end the EtherCAT master (Beckhoff TwinCAT) parameterizes each EtherCAT slave during the start-up phase to define which process data (size in bits/bytes, source location, transmission type) it wants to transfer to or from this slave. Incorrect configuration can prevent successful start-up of the slave.

For Beckhoff EtherCAT EL/ES slaves the following applies in general:

- The input/output process data supported by the device are defined by the manufacturer in the ESI/XML description. The TwinCAT EtherCAT Master uses the ESI description to configure the slave correctly.
- The process data can be modified in the system manager. See the device documentation.
  Examples of modifications include: mask out a channel, displaying additional cyclic information, 16-bit display instead of 8-bit data size, etc.
- In so-called “intelligent” EtherCAT devices the process data information is also stored in the CoE directory. Any changes in the CoE directory that lead to different PDO settings prevent successful startup of the slave. It is not advisable to deviate from the designated process data, because the device firmware (if available) is adapted to these PDO combinations.

If the device documentation allows modification of process data, proceed as follows (see Figure *“Configuring the process data”*).

- A: select the device to configure
- B: in the “Process Data” tab select Input or Output under SyncManager (C)
- D: the PDOs can be selected or deselected
- H: the new process data are visible as linkable variables in the system manager
  The new process data are active once the configuration has been activated and TwinCAT has been restarted (or the EtherCAT master has been restarted)
- E: if a slave supports this, Input and Output PDO can be modified simultaneously by selecting a so-called PDO record (“predefined PDO settings”).

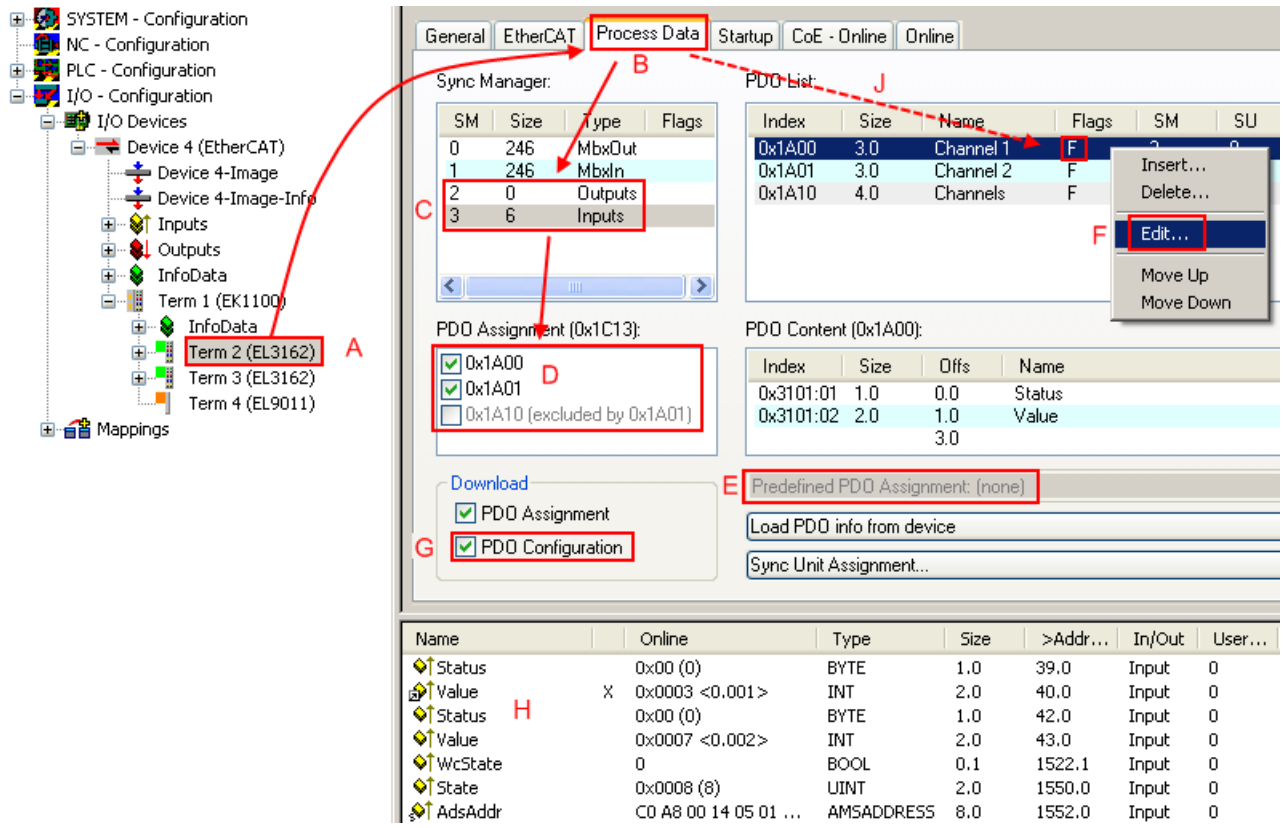


Fig. 79: *Configuring the process data*

Manual modification of the process data

According to the ESI description, a PDO can be identified as “fixed” with the flag “F” in the PDO overview (Fig. *“Configuring the process data”*, J). The configuration of such PDOs cannot be changed, even if TwinCAT offers the associated dialog (“Edit”). In particular, CoE content cannot be displayed as cyclic process data.This generally also applies in cases where a device supports download of the PDO configuration, “G”.In case of incorrect configuration the EtherCAT slave usually refuses to start and change to OP state. The System Manager displays an “invalid SM cfg” logger message:This error message (“invalid SM IN cfg” or “invalid SM OUT cfg”) also indicates the reason for the failed start.

5.2 General Notes - EtherCAT Slave Application

This summary briefly deals with a number of aspects of EtherCAT Slave operation under TwinCAT. More detailed information on this may be found in the corresponding sections of, for instance, the EtherCAT System Documentation.

Diagnosis in real time: WorkingCounter, EtherCAT State and Status

Generally speaking an EtherCAT Slave provides a variety of diagnostic information that can be used by the controlling task.

This diagnostic information relates to differing levels of communication. It therefore has a variety of sources, and is also updated at various times.

Any application that relies on I/O data from a fieldbus being correct and up to date must make diagnostic access to the corresponding underlying layers. EtherCAT and the TwinCAT System Manager offer comprehensive diagnostic elements of this kind. Those diagnostic elements that are helpful to the controlling task for diagnosis that is accurate for the current cycle when in operation (not during commissioning) are discussed below.

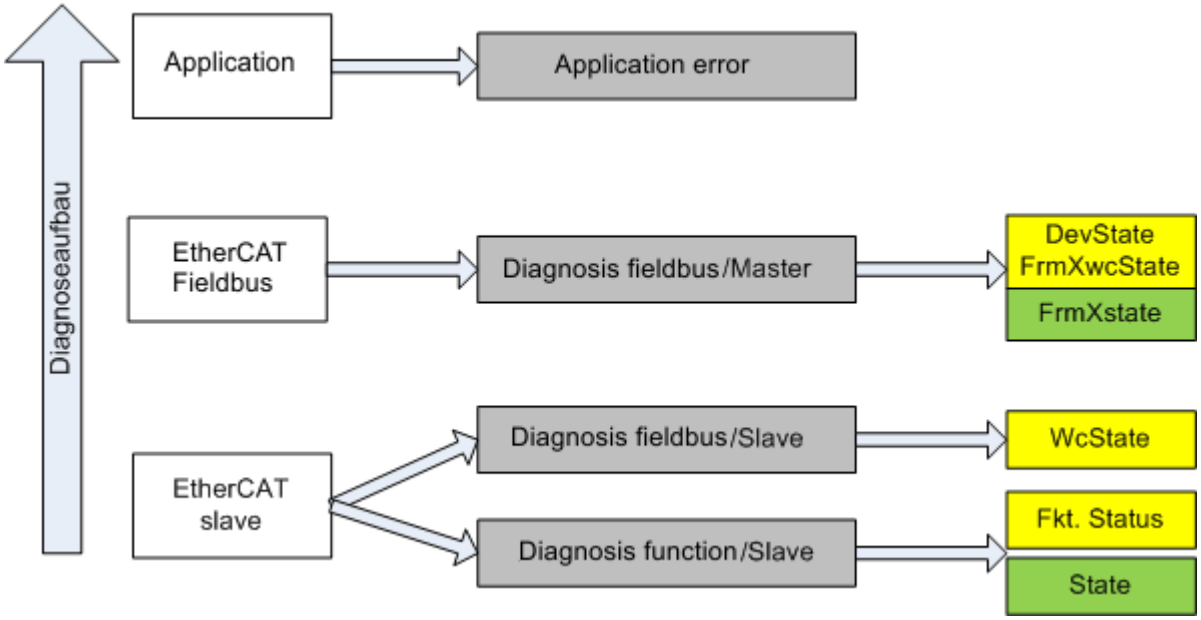


Fig. 80: Selection of the diagnostic information of an EtherCAT Slave

In general, an EtherCAT Slave offers

- communication diagnosis typical for a slave (diagnosis of successful participation in the exchange of process data, and correct operating mode)
  This diagnosis is the same for all slaves.

  as well as
- function diagnosis typical for a channel (device-dependent)
  See the corresponding device documentation

The colors in Fig. *"Selection of the diagnostic information of an EtherCAT Slave"* also correspond to the variable colors in the System Manager, see Fig. *"Basic EtherCAT Slave Diagnosis in the PLC"*.

Colour	Meaning
yellow	Input variables from the Slave to the EtherCAT Master, updated in every cycle
red	Output variables from the Slave to the EtherCAT Master, updated in every cycle
green	Information variables for the EtherCAT Master that are updated acyclically. This means that it is possible that in any particular cycle they do not represent the latest possible status. It is therefore useful to read such variables through ADS.

Fig. *"Basic EtherCAT Slave Diagnosis in the PLC"* shows an example of an implementation of basic EtherCAT Slave Diagnosis. A Beckhoff EL3102 (2-channel analogue input terminal) is used here, as it offers both the communication diagnosis typical of a slave and the functional diagnosis that is specific to a channel. Structures are created as input variables in the PLC, each corresponding to the process image.

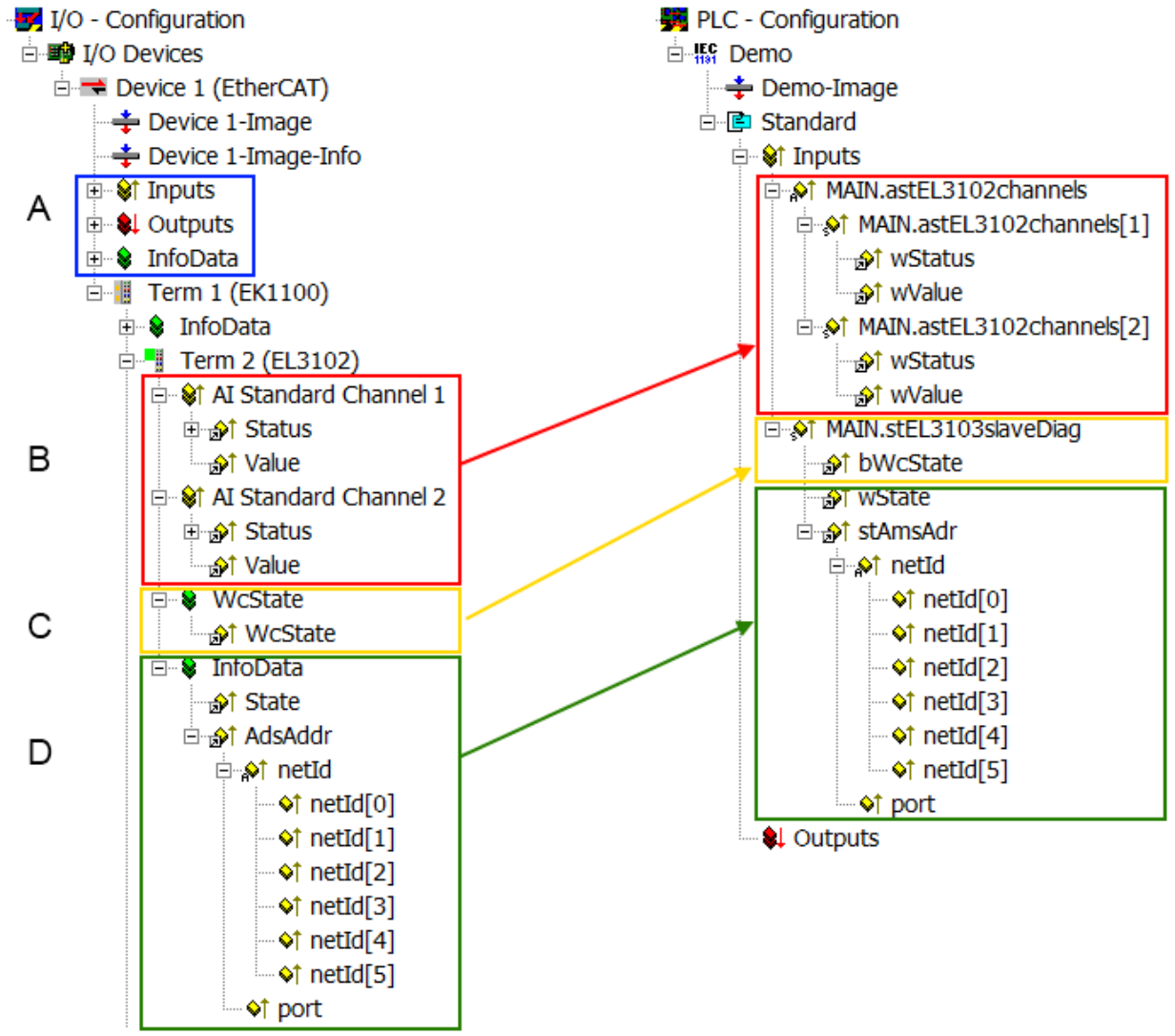


Fig. 81: Basic EtherCAT Slave Diagnosis in the PLC

The following aspects are covered here:

Code	Function	Implementation	Application/evaluation
A	The EtherCAT Master's diagnostic information updated acyclically (yellow) or provided acyclically (green).		At least the DevState is to be evaluated for the most recent cycle in the PLC. The EtherCAT Master's diagnostic information offers many more possibilities than are treated in the EtherCAT System Documentation. A few keywords: • CoE in the Master for communication with/through the Slaves • Functions from *TcEtherCAT.lib* • Perform an OnlineScan
B	In the example chosen (EL3102) the EL3102 comprises two analogue input channels that transmit a single function status for the most recent cycle.	Status • the bit significations may be found in the device documentation • other devices may supply more information, or none that is typical of a slave	In order for the higher-level PLC task (or corresponding control applications) to be able to rely on correct data, the function status must be evaluated there. Such information is therefore provided with the process data for the most recent cycle.
C	For every EtherCAT Slave that has cyclic process data, the Master displays, using what is known as a WorkingCounter, whether the slave is participating successfully and without error in the cyclic exchange of process data. This important, elementary information is therefore provided for the most recent cycle in the System Manager 1. at the EtherCAT Slave, and, with identical contents 2. as a collective variable at the EtherCAT Master (see Point A) for linking.	WcState (Working Counter) 0: valid real-time communication in the last cycle 1: invalid real-time communication This may possibly have effects on the process data of other Slaves that are located in the same SyncUnit	In order for the higher-level PLC task (or corresponding control applications) to be able to rely on correct data, the communication status of the EtherCAT Slave must be evaluated there. Such information is therefore provided with the process data for the most recent cycle.
D	Diagnostic information of the EtherCAT Master which, while it is represented at the slave for linking, is actually determined by the Master for the Slave concerned and represented there. This information cannot be characterized as real-time, because it • is only rarely/never changed, except when the system starts up • is itself determined acyclically (e.g. EtherCAT Status)	State current Status (INIT..OP) of the Slave. The Slave must be in OP (=8) when operating normally. *AdsAddr* The ADS address is useful for communicating from the PLC/task via ADS with the EtherCAT Slave, e.g. for reading/writing to the CoE. The AMS-NetID of a slave corresponds to the AMS-NetID of the EtherCAT Master; communication with the individual Slave is possible via the *port* (= EtherCAT address).	Information variables for the EtherCAT Master that are updated acyclically. This means that it is possible that in any particular cycle they do not represent the latest possible status. It is therefore possible to read such variables through ADS.

NOTE

Diagnostic information

It is strongly recommended that the diagnostic information made available is evaluated so that the application can react accordingly.

CoE Parameter Directory

The CoE parameter directory (CanOpen-over-EtherCAT) is used to manage the set values for the slave concerned. Changes may, in some circumstances, have to be made here when commissioning a relatively complex EtherCAT Slave. It can be accessed through the TwinCAT System Manager, see Fig. *"EL3102, CoE directory"*:

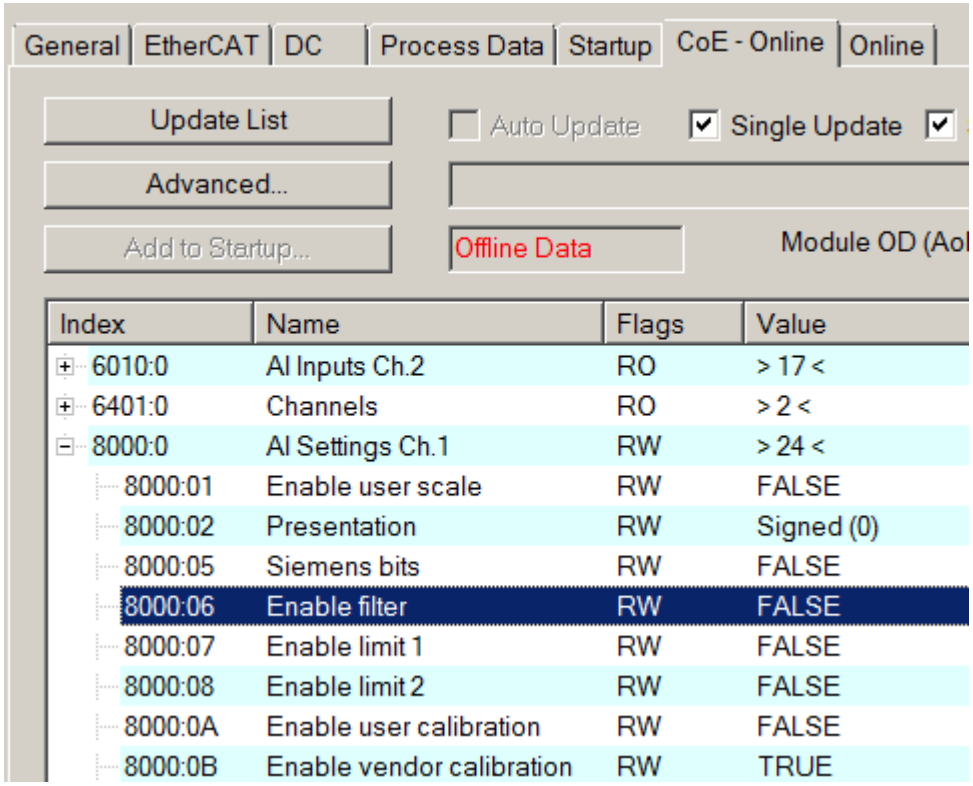


Fig. 82: EL3102, CoE directory

EtherCAT System Documentation

The comprehensive description in the EtherCAT System Documentation (EtherCAT Basics --> CoE Interface) must be observed!

A few brief extracts:

- Whether changes in the online directory are saved locally in the slave depends on the device. EL terminals (except the EL66xx) are able to save in this way.
- The user must manage the changes to the StartUp list.

Commissioning aid in the TwinCAT System Manager

Commissioning interfaces are being introduced as part of an ongoing process for EL/EP EtherCAT devices. These are available in TwinCAT System Managers from TwinCAT 2.11R2 and above. They are integrated into the System Manager through appropriately extended ESI configuration files.

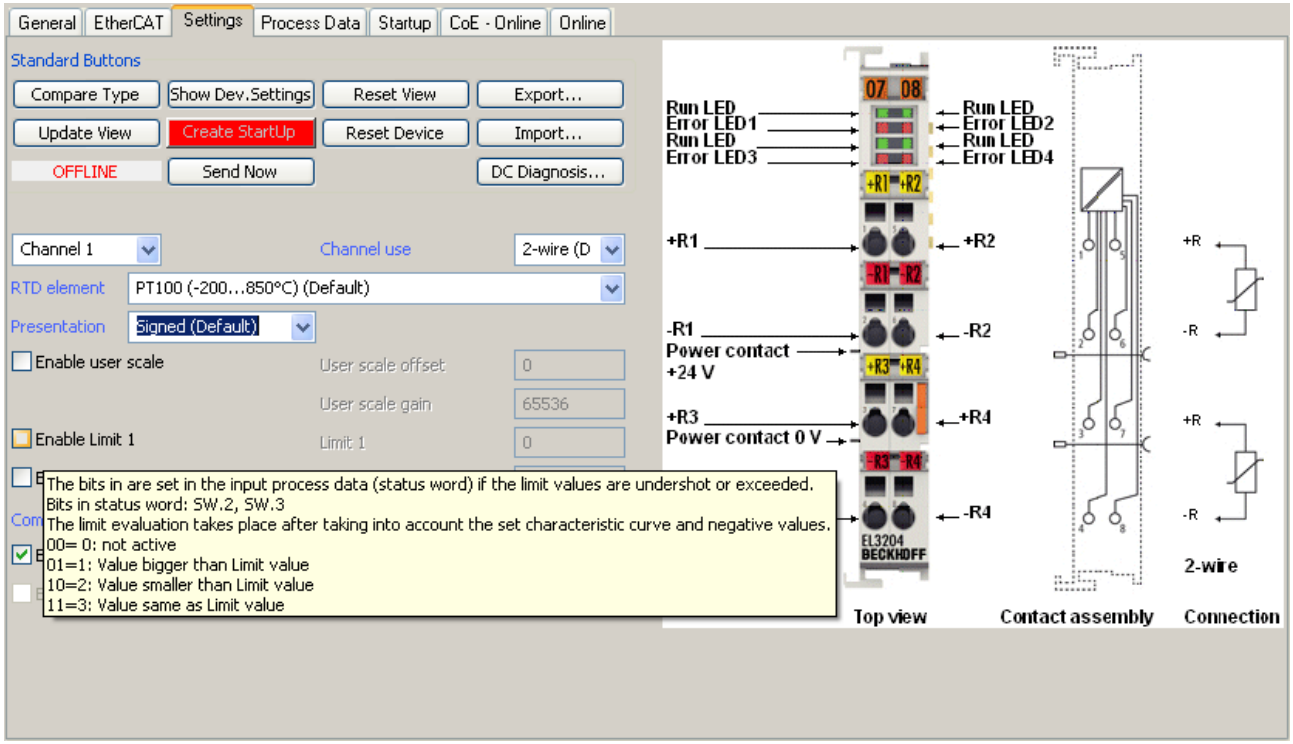


Fig. 83: Example of commissioning aid for a EL3204

This commissioning process simultaneously manages

- CoE Parameter Directory
- DC/FreeRun mode
- the available process data records (PDO)

Although the "Process Data", "DC", "Startup" and "CoE-Online" that used to be necessary for this are still displayed, it is recommended that, if the commissioning aid is used, the automatically generated settings are not changed by it.

The commissioning tool does not cover every possible application of an EL/EP device. If the available setting options are not adequate, the user can make the DC, PDO and CoE settings manually, as in the past.

EtherCAT State: automatic default behaviour of the TwinCAT System Manager and manual operation

After the operating power is switched on, an EtherCAT Slave must go through the following statuses

- INIT
- PREOP
- SAFEOP
- OP

to ensure sound operation. The EtherCAT Master directs these statuses in accordance with the initialization routines that are defined for commissioning the device by the ES/XML and user settings (Distributed Clocks (DC), PDO, CoE). See also the section on "Principles of Communication, EtherCAT State Machine [▶ 34]" in this connection. Depending how much configuration has to be done, and on the overall communication, booting can take up to a few seconds.

The EtherCAT Master itself must go through these routines when starting, until it has reached at least the OP target state.

The target state wanted by the user, and which is brought about automatically at start-up by TwinCAT, can be set in the System Manager. As soon as TwinCAT reaches the status RUN, the TwinCAT EtherCAT Master will approach the target states.

Standard setting

The advanced settings of the EtherCAT Master are set as standard:

- EtherCAT Master: OP
- Slaves: OP
  This setting applies equally to all Slaves.

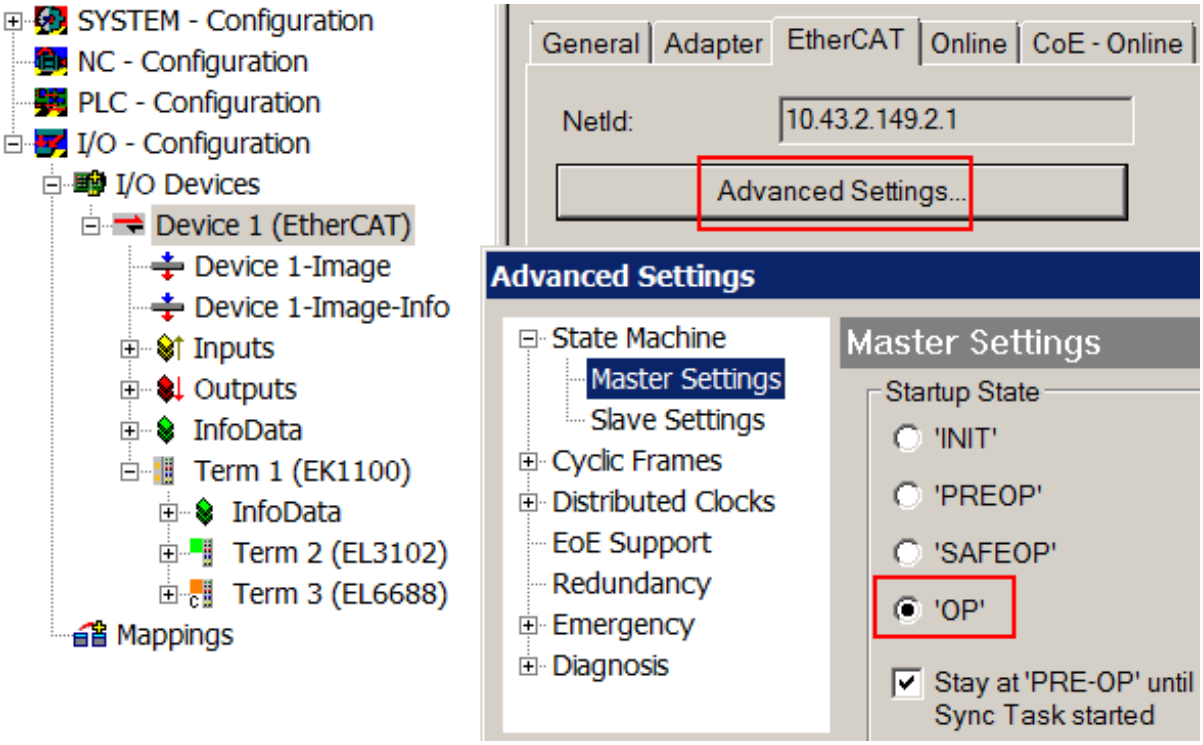


Fig. 84: Default behaviour of the System Manager

In addition, the target state of any particular Slave can be set in the "Advanced Settings" dialogue; the standard setting is again OP.

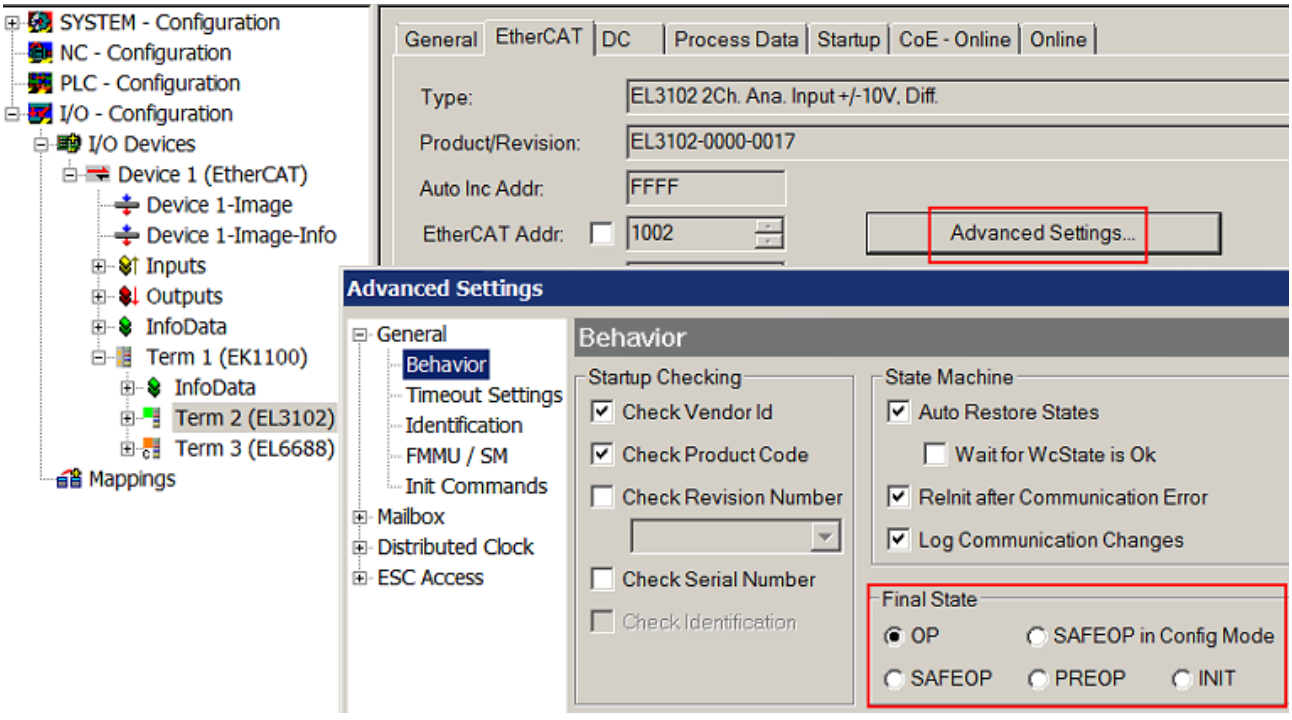


Fig. 85: Default target state in the Slave

Manual Control

There are particular reasons why it may be appropriate to control the states from the application/task/PLC. For instance:

- for diagnostic reasons
- to induce a controlled restart of axes
- because a change in the times involved in starting is desirable

In that case it is appropriate in the PLC application to use the PLC function blocks from the *TcEtherCAT.lib*, which is available as standard, and to work through the states in a controlled manner using, for instance, *FB_EcSetMasterState*.

It is then useful to put the settings in the EtherCAT Master to INIT for master and slave.

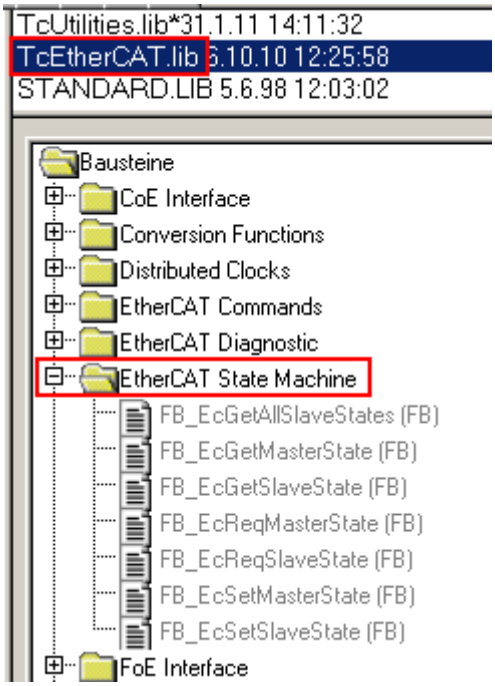
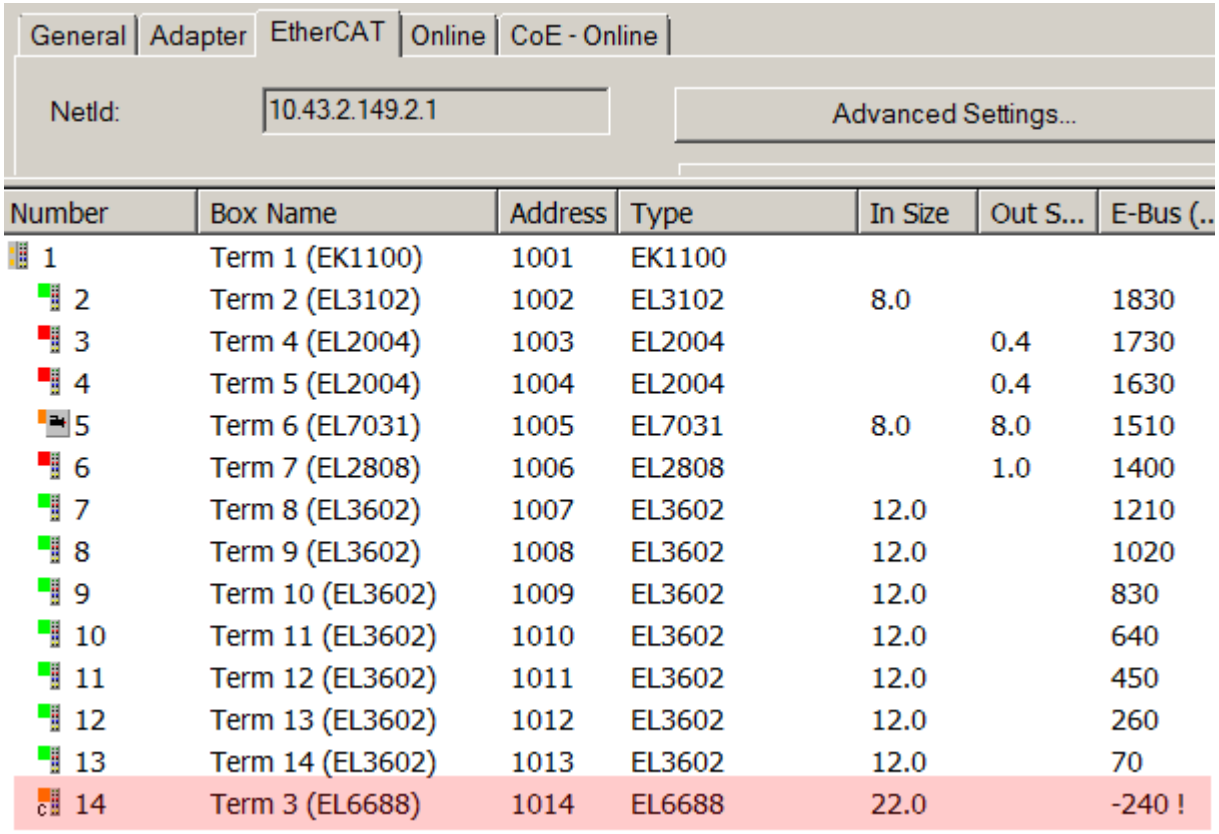


Fig. 86: PLC function blocks

Note regarding E-Bus current

EL/ES terminals are placed on the DIN rail at a coupler on the terminal strand. A Bus Coupler can supply the EL terminals added to it with the E-bus system voltage of 5 V; a coupler is thereby loadable up to 2 A as a rule. Information on how much current each EL terminal requires from the E-bus supply is available online and in the catalogue. If the added terminals require more current than the coupler can supply, then power feed terminals (e.g. EL9410) must be inserted at appropriate places in the terminal strand.

The pre-calculated theoretical maximum E-Bus current is displayed in the TwinCAT System Manager as a column value. A shortfall is marked by a negative total amount and an exclamation mark; a power feed terminal is to be placed before such a position.



Number	Box Name	Address	Type	In Size	Out S...	E-Bus (..
1	Term 1 (EK1100)	1001	EK1100			
2	Term 2 (EL3102)	1002	EL3102	8.0		1830
3	Term 4 (EL2004)	1003	EL2004		0.4	1730
4	Term 5 (EL2004)	1004	EL2004		0.4	1630
5	Term 6 (EL7031)	1005	EL7031	8.0	8.0	1510
6	Term 7 (EL2808)	1006	EL2808		1.0	1400
7	Term 8 (EL3602)	1007	EL3602	12.0		1210
8	Term 9 (EL3602)	1008	EL3602	12.0		1020
9	Term 10 (EL3602)	1009	EL3602	12.0		830
10	Term 11 (EL3602)	1010	EL3602	12.0		640
11	Term 12 (EL3602)	1011	EL3602	12.0		450
12	Term 13 (EL3602)	1012	EL3602	12.0		260
13	Term 14 (EL3602)	1013	EL3602	12.0		70
14	Term 3 (EL6688)	1014	EL6688	22.0		-240 !

Fig. 87: Illegally exceeding the E-Bus current

From TwinCAT 2.11 and above, a warning message "E-Bus Power of Terminal..." is output in the logger window when such a configuration is activated:

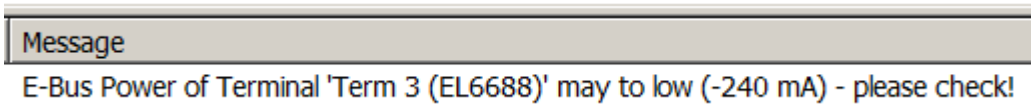


Fig. 88: Warning message for exceeding E-Bus current

NOTE

Caution! Malfunction possible!

The same ground potential must be used for the E-Bus supply of all EtherCAT terminals in a terminal block!

5.3 TwinCAT (2.1x) System Manager

5.3.1 Configuration by means of the TwinCAT System Manager

The TwinCAT System Manager tool is used for the configuration of the EL6751 CANopen master/slave terminal. The System Manager provides a representation of the number of programs of the TwinCat PLC systems, the configuration of the axis control and of the connected I/O channels as a structure, and organizes the mapping of the data traffic.

Fig. 89: TwinCAT System Manager logo

For applications without TwinCAT PLC or NC, the TwinCAT System Manager Tool configures the programming interfaces for a wide range of application programs:

- ActiveX control (ADS-OCX) for e.g. Visual Basic, Visual C++, Delphi, etc.
- DLL interface (ADS-DLL) for e.g. Visual C++ projects
- Script interface (ADS script DLL) for e.g. VBScript, JScript, etc.

System Manager – Features

- Bit-wise association of server process images and I/O channels
- Standard data formats such as arrays and structures
- User defined data formats
- Continuous variable linking
- Drag and Drop
- Import and export at all levels

The procedure, and the configuration facilities in the System Manager are described below.

- EL6751 - CANopen master terminal [▶ 76]
- CAN interface [▶ 79]
- EL6751-0010 - CANopen slave terminal [▶ 82]

EL6751 - CANopen master terminal

Context menu

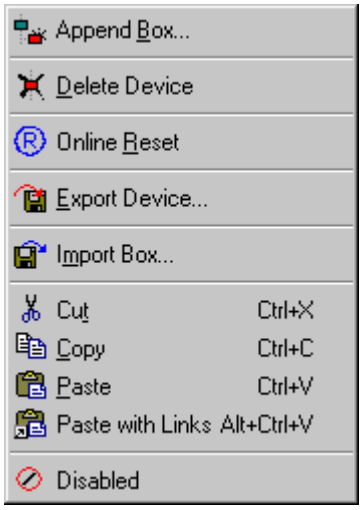


Fig. 90: Add Box... <Insert>

Adds CANopen slaves (boxes). The following boxes are currently supported (more detailed description of the boxes is given further down):

Supported boxes	Description
BK5110	Economy Bus Coupler
BK5120	Economy + Bus Coupler
LC5100	Low-Cost Bus Coupler
BK5150	Compact Bus Coupler
BK5151	Compact Bus Coupler with D-Sub connection
BC5150	Compact Bus Terminal Controller with 48 kbyte program memory
BX5100	BX Bus Terminal Controller with 256 kbyte program memory
IPxxxx-B510	Fieldbus compact box: CANopen in/output module, protection class IP67
CANopen Node [▶ 88]	General CANopen device or general CAN device (access via CAN layer 2)

Delete Device... <Del>

Removes the EL6751 and all subordinated elements from the I/O configuration.

Online Reset

Initiates an online reset on the CANopen bus.

"EL6751" tab

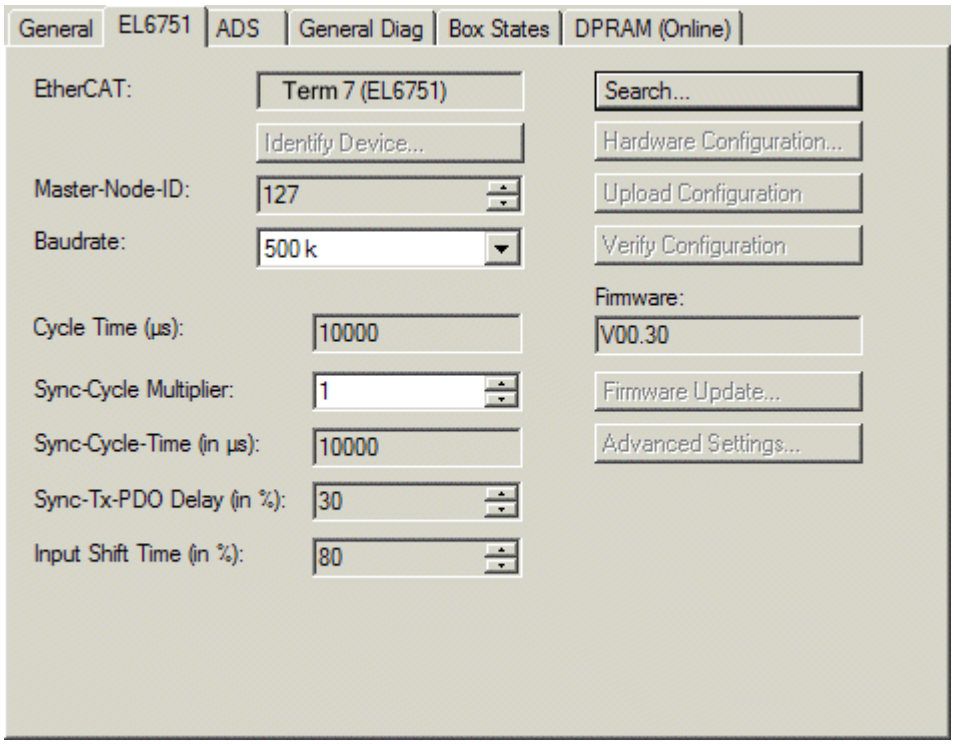


Fig. 91: EL6751 tab

EtherCAT

Terminal ID in the terminal network.

Master Node ID

Node address of the EL6751. Value range: 1...127. Determines the identifier of the master heartbeat telegram. Ensure that it is not the same as a slave node address.

Baud rate

Set the baud rate [▶ 109] here. Automatically tests whether the connected slave also supports this baud rate.

Cycle time

Displays the cycle time of the corresponding highest priority task. The display is updated when the mapping is generated.

Sync-Cycle Multiplier

CANopen SYNC Cycle Time = (Task) Cycle Time x Sync-Cycle Multiplier. Event driven PDO communications and cyclical synchronized PDO communication are frequently combined when used in conjunction with CANopen. In order to be able to respond rapidly to an event, the TwinCAT task cycle time has to be less than the CANopen SYNC cycle time. If the sync cycle multiplier is set to values > 1, the TwinCAT task is called repeatedly before the SYNC telegram is sent again.

Sync-Cycle Time

The cycle time of the CANopen snyc telegram is displayed here. It results from the cycle time of the highest-priority task, whose process data are linked with the card, and from the sync cycle multiplier.

Sync-Tx-PDO Delay

Directly after the SYNC telegram, the synchronized slaves send their input data/actual values. The EL6751 can delay the transmission of the output data or setpoint values (TxPDOs from the point of view of the terminal) in order to minimize the telegram burst directly after the SYNC. This delay is set in percent of the SYNC cycle time with the parameter Sync-Tx-PDO -Delay.

Sample:

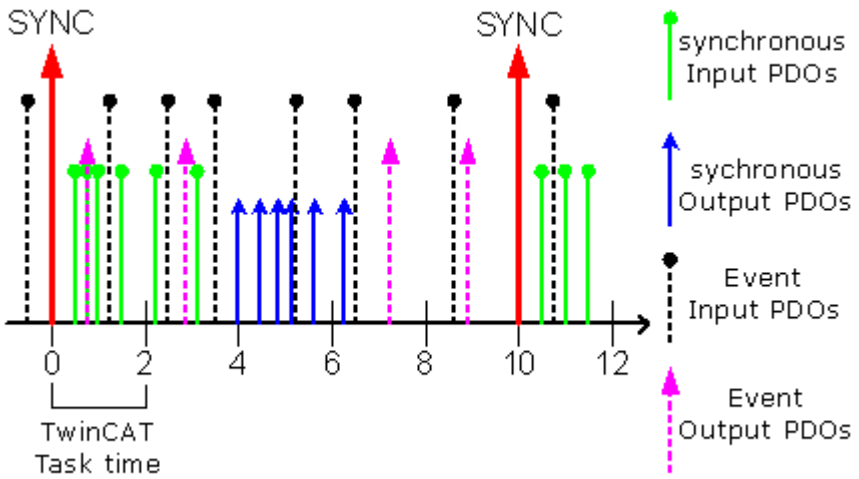


Fig. 92: Diagram: Sync-Tx-PDO-Delay sample

Task Cycle Time = 2000µs, Sync Cycle Multiplier = 5, Sync Tx-PDO Delay =40. Event-controlled PDOs can be processed by the PLC task every 2 ms; the CANopen sync cycle is 10 ms; the EL6751 transmits its synchronous PDOs 4 ms (= 40% of 10 ms) after the SYNC.

Input Shift Time [EL6751 only]

Specifies the fraction of the EtherCAT cycle (in %) after which the inputs in the EtherCAT slave controller are updated. The later this is the case, the shorter the dead time from receipt of a TxPDO until the time when the associated input data of the task are available. The minimum time to be maintained between the start of the input update and the next EtherCAT cycle is the CalcAndCopy time (0x1C33:06), which depends on the number of configured CAN slaves and can be measured in OPERATIONAL state (set entry 0x1C32:08 to 1, then read entry 0x1C33:06).

Search...

This function searches for all existing channels of the EL6751 and the desired one can be selected.

Hardware Configuration... [FC510x only]

In which the address is set in the lower memory area (below 1 MB) of the PC.

Upload Configuration

This function scans the CANopen network and all devices found are added to the EL6751 (no box may be appended). In the case of Beckhoff boxes, reads the configuration precisely. In the case of external devices, the PDO configuration and the identity object are read and evaluated. This function is possible only in Config mode.

Verify Configuration

Allows a comparison of the expected (entered) network configuration with the devices actually found in the network. The data from the CANopen Identify Object are read and compared. In the case of Beckhoff boxes the connected Bus Terminals or extension modules are read and compared (under preparation). This function is possible only in Config mode.

Firmware

Shows the current firmware version of the EL6751.

Firmware Update... [FC510x only]

The firmware update is carried out via the associated hardware.

"ADS" tab

The EL6751 is an ADS device with its own net ID, which can be changed here. All ADS services (diagnostics, acyclical communication) associated with the EL6751 device must address the card via this NetID.

"Box States" tab

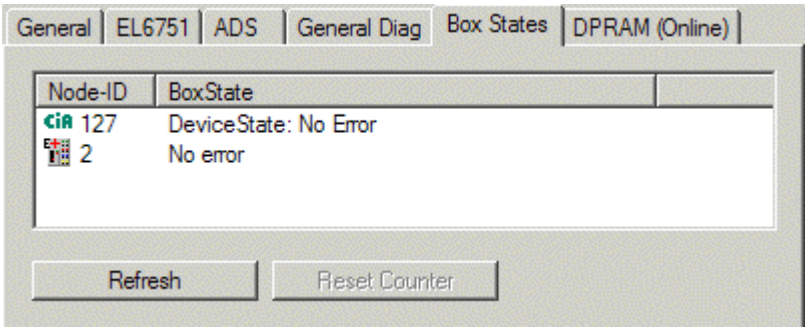


Fig. 93: "Box States" tab

Displays an overview of all current box statuses.

"(Online) DPRAM" tab

Refer to "Online display of DPRAM" in the System Manager documentation

CAN interface

Insert the "CAN interface" box directly behind the EL6151 device (see fig. Selection dialog "Inserting a box")

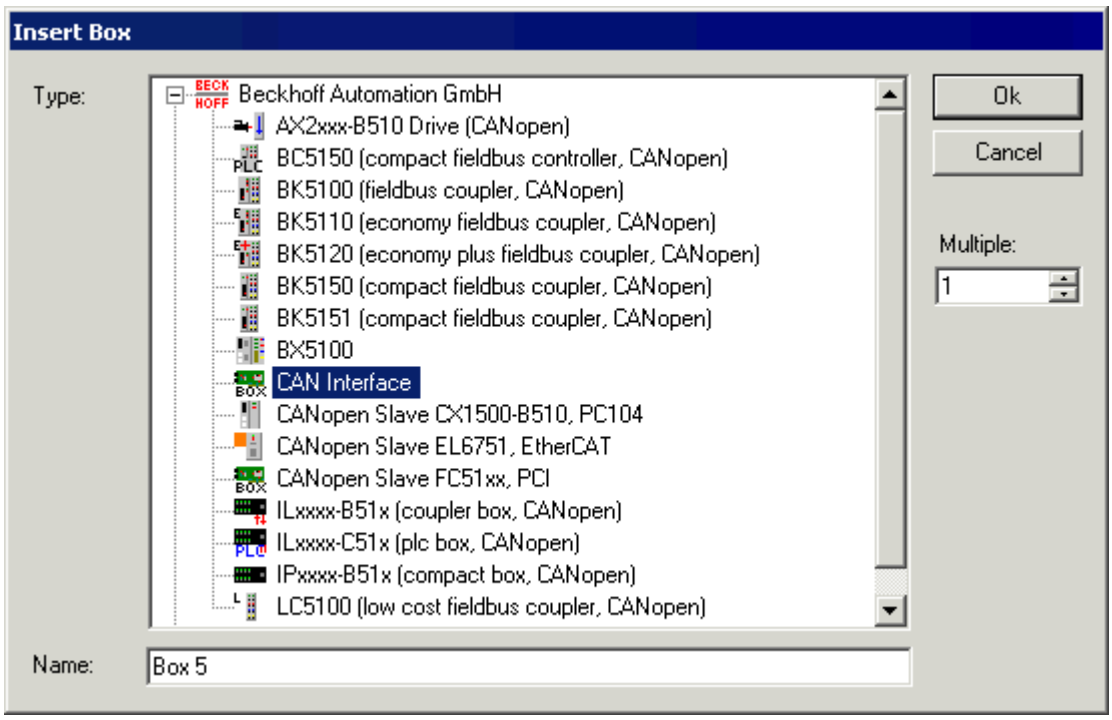


Fig. 94: Selection dialog "Inserting a box"

After the box has been inserted, the following dialog appears for the configuration of the CAN interface:

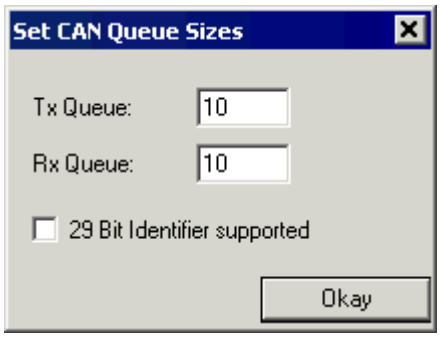


Fig. 95: Can Queue Sizes setting

Tx queue and Rx queue define the number of messages that are exchanged between the task and the CANopen master in a task cycle. If the message queues are to transmit 29-bit identifiers in addition, activate the checkbox "29 Bit Identifier supported".

The process image of the CAN interface then looks like this:

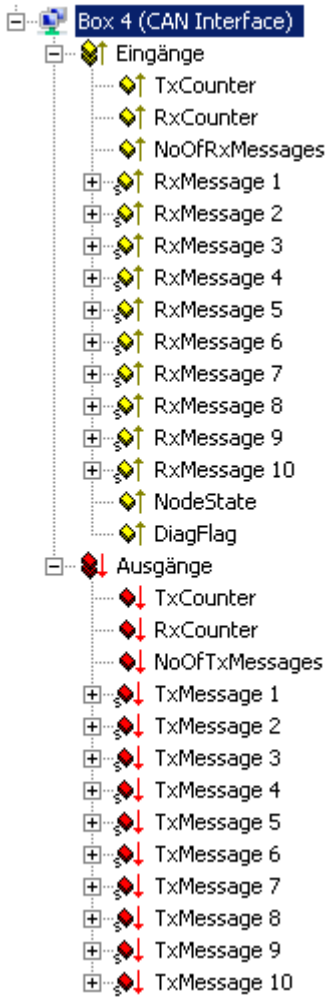


Fig. 96: CAN interface process image

Message structure with 29-bit support

- Length (0..8)
  - CobId
    o Bit 0-28: 29 Bit-Identifier
    o Bit 30: RTR
    o Bit 31: 0: normal Message (11 Bit Identifier), 1: extended Message (29 Bit-Identifier)
  - Data[8]

Message structure without 29-bit support

- CobId
  o Bit 0-3: Length (0..8)
  o Bit 4: RTR
  o Bit 5-15: 11 Bit-Identifier
- Data[8]

The "CAN Rx Filter" tab of the CAN interface box in the TwinCAT tree is used for the setting of the filter for the Rx messages (default: all messages are received).
After clicking the "Append..." button, the following dialog appears:

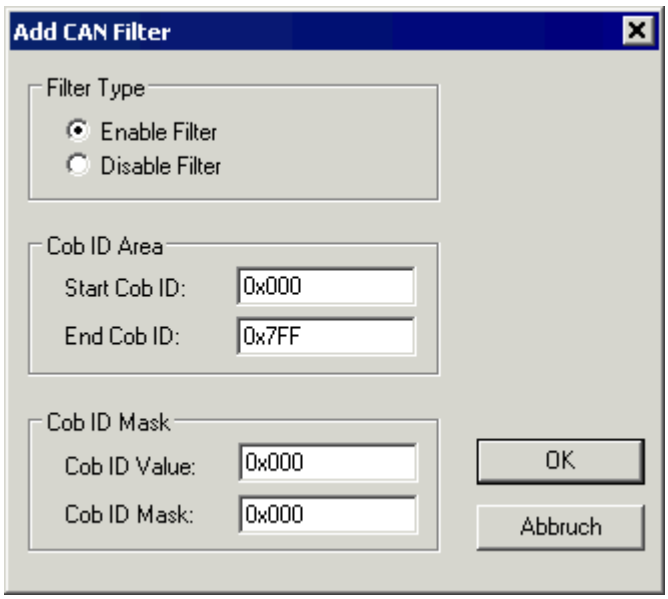


Fig. 97: Dialog „Add CAN Filter“

An "Enable Filter" defines an area or a mask of COB-ID that is received; a "Disable Filter" defines an area or a mask of COB-ID that is not received.

Sample code: Sending messages from the PLC

```
if Outputs.TxCounter = Inputs.TxCounter then
          for i=0 to NumberOfMessagesToSend do
                  Outputs.TxMessage[i] = MessageToSend[i];
          End_for
          Outputs.NoOfTxMessages = NumberOfMessagesToSend;
          Outputs.TxCounter := Outputs.TxCounter + 1;
end_if
```

Sample code: Receiving messages from the PLC

```
if Outputs.RxCounter <> Inputs.RxCounter then
          for I := 0 to (Inputs.NoOfRxMessages-1) do
                   MessageReceived[i] := Inputs.RxMessage [i];
          End_for
          Outputs.RxCounter := Outputs.RxCounter+1;
end_if
```

EL6751-0010 - CANopen slave terminal

In the system configuration tree structure right-click on I/O Devices and "Append Device" to open the selection list of supported fieldbus cards:

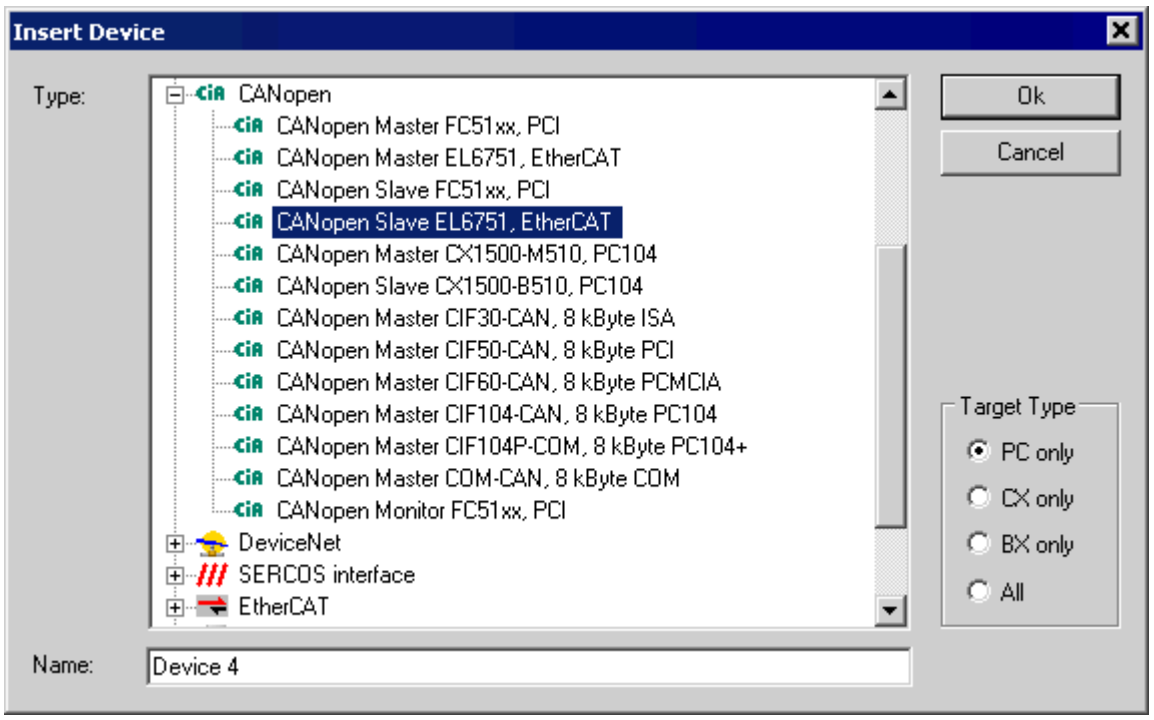


Fig. 98: EL6751-0010: Dialog "Appending an I/O device"

Select EL6751-0010 CANopenSlave. TwinCAT searches for the terminal and displays the memory addresses and slots it finds. Select the required address and confirm.

I/O Device EL6751-0010 CANopen Slave

Selecting the inserted I/O device in the tree structure opens a dialog with different configuration options:

"EL6751-0010" tab

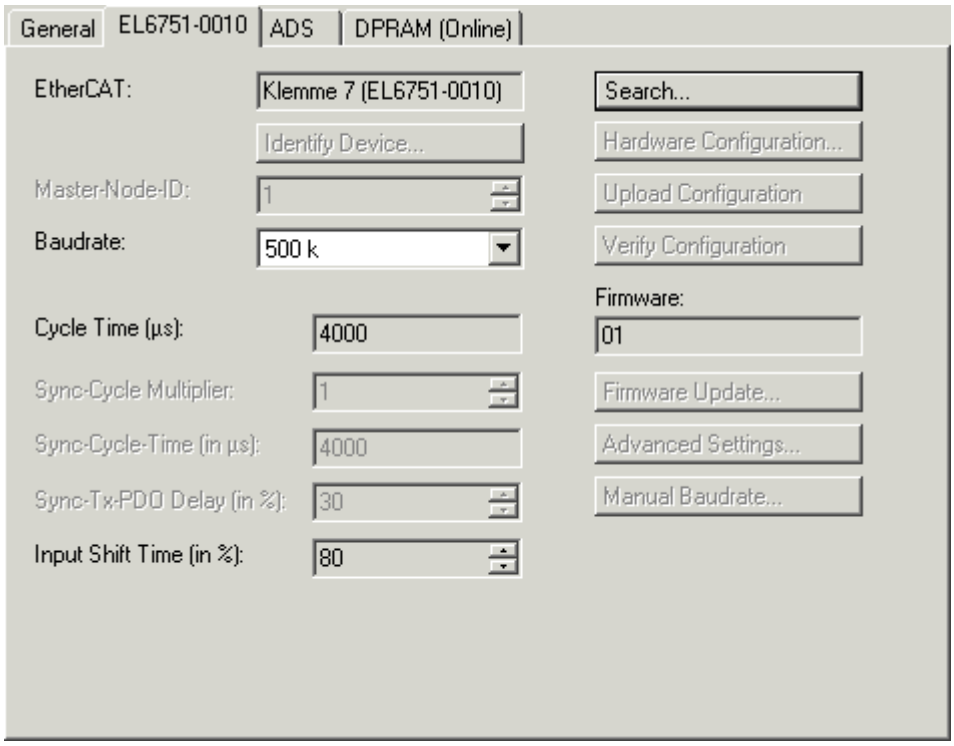


Fig. 99: "EL6751-0010" tab

EtherCAT

Terminal ID in the terminal network.

Baud rate

Set the baud rate [▶ 109] here.

Cycle Time

Displays the cycle time of the corresponding highest priority task. The display is updated when the mapping is generated. The network variables are updated with the cycle of this task.

Search...

Searches for all available EL6751-0010 channels, from which the required channel can be selected. In the case of an FC5102 both channels A and B appear. These behave in logical terms like two FC5101 cards.

Firmware

Displays the current EL6751-0010 firmware version.

Firmware Update... [FC510x only]

The firmware update for the EL6751-0010 is carried out via the associated EL6751-0010 terminal.

"ADS" tab

The EL6751-0010 is an ADS device with its own net ID, which can be changed here. All ADS services (diagnostics, acyclic communication) associated with the EL6751-0010 device must address the card via this NetID. Additional ADS Net IDs can be entered for addressing subordinate ADS devices (e.g. an additional fieldbus card in the same PC) via the card.

"(Online) DPRAM" tab

Read access to the DPRAM of the card is provided for diagnostic purposes.

EL6751-0010 Slave box

An "EL6781-0010 (CANopen Slave)" box is created automatically. Further parameters have to be set:

CAN Node tab:

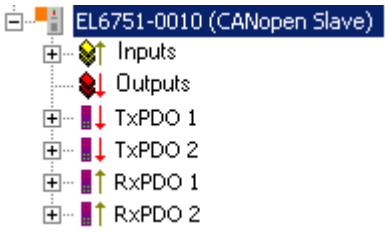


Fig. 100: EL6751-0010 TwinCAT tree

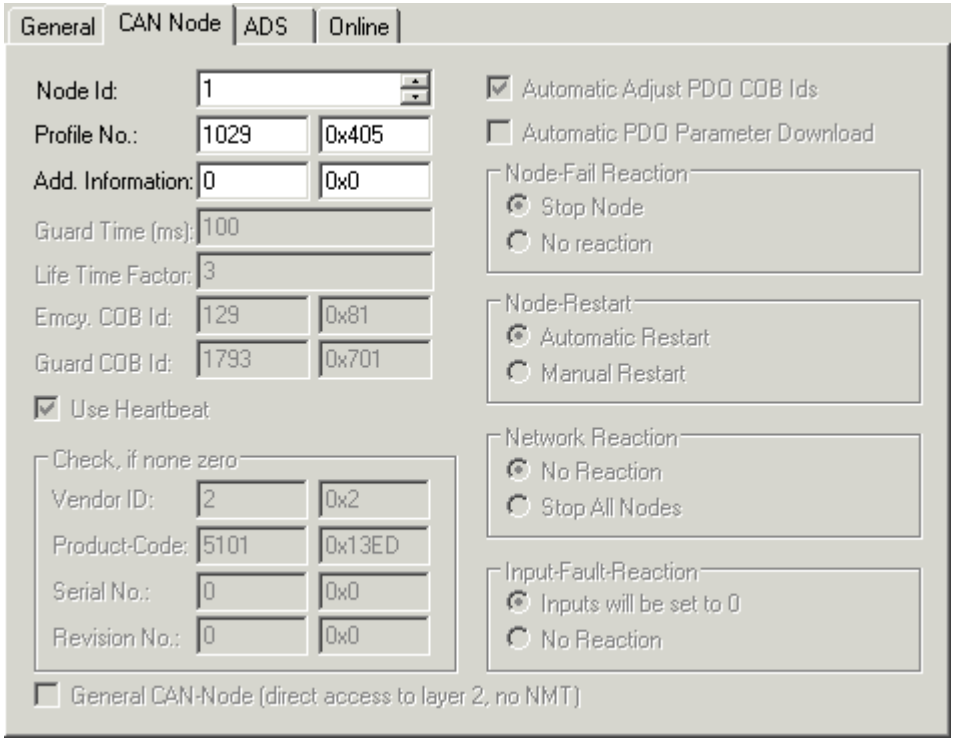


Fig. 101: "CAN node" tab

Node ID:

Set the EL6751-0010 node address.

The profile number and the Add. Information form the DeviceType that can be read via object 0x1000.

Configuring network variables

PLC variables communicated by the EL6751-0010 device are referred to as network variables. These variables must be created and added to the associated PDOs. To this end right-click on the PDO and select "Inserting variables". The following dialog box opens:

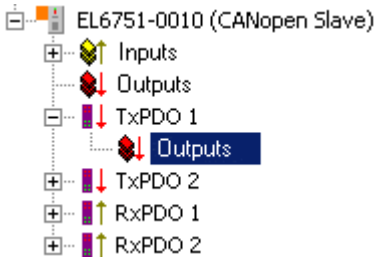


Fig. 102: EL6751-0010 TwinCAT tree, outputs

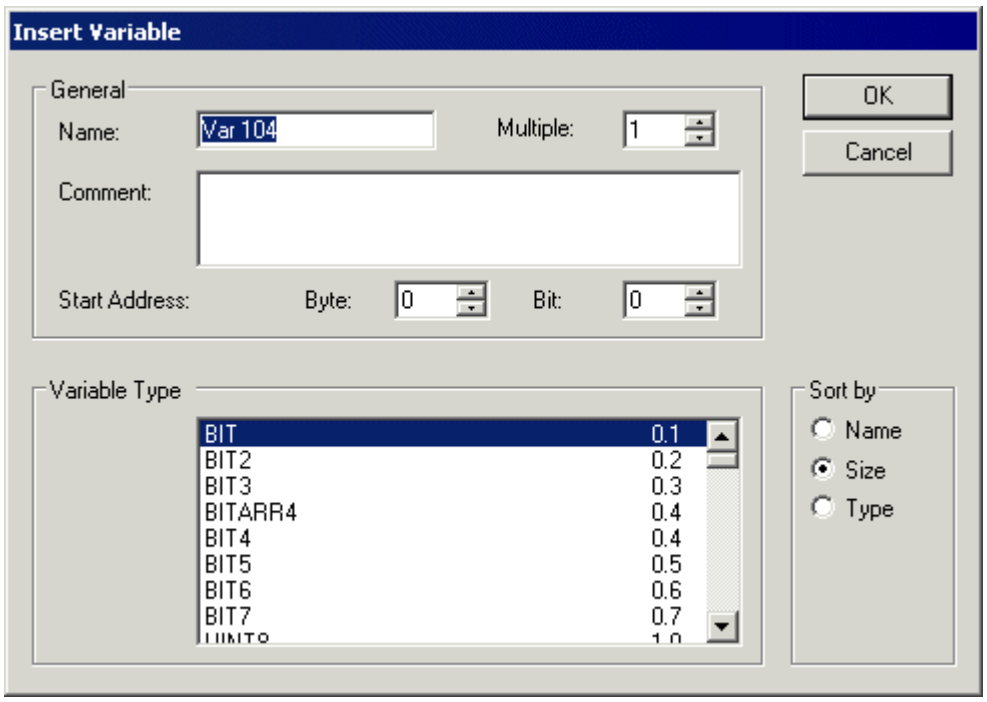


Fig. 103: Dialog "Inserting variables"

If several variables of the same type are added at the same time ("multiple"), the start address for the data is calculated automatically within the PDO. If individual variables are added the start address must be specified explicitly.

The network variables added in this way can then be linked in the familiar way (see System Manager documentation in the TwinCAT Information System) with the variables for the different tasks.

5.3.2 BECKHOFF CANopen Bus Coupler

The BK51xx Bus Coupler and the IPxxx-B510 Fieldbus Box are installed in the **CANopen** bus. The specific properties which distinguish them from other Bus Couplers and/or fieldbus box modules are then described below.

Types	Description
BK5110	Economy Bus Coupler
BK5120	Economy + Bus Coupler
LC5100	Low-Cost Bus Coupler
BK5150	Compact Bus Coupler
BK5151	Compact Bus Coupler with D-Sub connection
BC5150	Compact Bus Terminal Controller with 48 kbyte program memory
BX5100	BX Bus Terminal Controller with 256 kbyte program memory
IPxxxx-B510	Fieldbus compact box: CANopen in/output module, protection class IP67

"BK51x0" tab

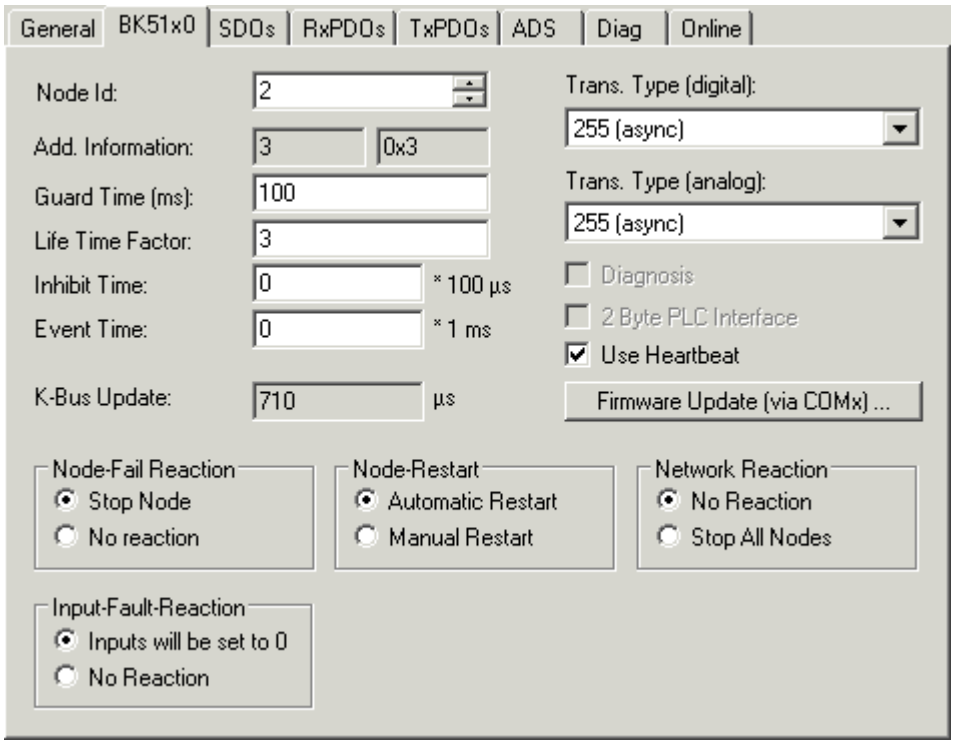


Fig. 104: "BK51x0" tab

Node Id
Sets the node ID of the CAN bus device (between 1 and 63 (BK51x0) and/or 1 and 99 (IPxxxx-B510)). This value must comply with the value set at the Bus Coupler and/or at the compact box.

Guard time
Cycle time for the node monitoring (node guarding).

Lifetime factor
Guard time multiplied produces the watchdog time for the monitoring of the master by the coupler (life guarding). Life guarding is deactivated if the lifetime factor is set to zero.

Inhibit Time
Displays the minimum send interval for PDOs (telegrams) with analog and special signals. If more than digital 64 signals are present, these are also provided with this Inhibit Time [▶ 101].

Event Time
Event timer for transmit PDOs. Expiry of this timer is treated as an additional event for the corresponding PDO, so that the PDO will then be transmitted. If the application event occurs during a timer period, it will also be transmitted, and the timer is reset.

K-Bus update
Calculates the anticipated duration of a complete update of the terminal bus (according to type and number of connected terminals).

Use Heartbeat
Heartbeat is used for monitoring of the node. If deactivated, the guarding is used for monitoring.

Trans. type
Gives the Transmission Type [▶ 101] for digital / analog input telegrams. 254 + 255 corresponds to the event-driven transfer, 1 ... 240 are synchronous transfer types. For further details see also BK51X0 manual.

Firmware Update
Enables the updating of the coupler firmware via the serial interface (requires KS2000 software package interface cable).

"SDOs" tab

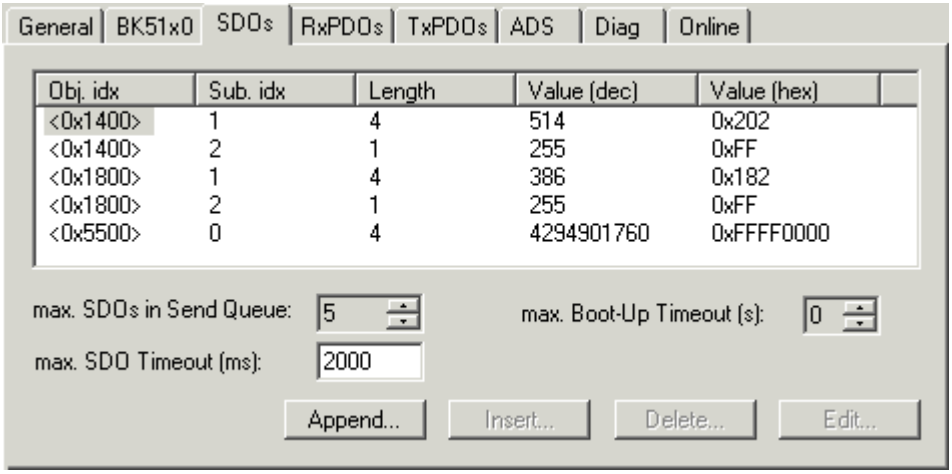


Fig. 105: "SDOs" tab

SDO inputs sent to the node at StartUp are displayed/managed on this page. Inputs with an object index in straight brackets are automatically created on the basis of the updated terminal configuration. Other inputs can be managed using ”Add", ”Insert", ”Delete" and ”Edit".

"RxPDOs" tab

Displays the RxPDOs used for the node.

"TxPDOs" tab

Displays the TxPDOs used for the node.

“ADS” tab

The node (Bus Coupler) is assigned an ADS port to enable writing and reading of SDO objects at runtime (e.g. from the PLC). It can be changed if required. The ADS IndexGroup contains the CANopen object index and the ADS IndexOffset contains the CANopen Sub-Index. See chapter SDO communication [▶ 114] for details of SDO communication via ADS.

“Diag” tab

Diagnostic information is displayed here. The window contents are not cyclically refreshed; select the "Refresh" button if necessary. The diagnostic information displayed can also be queried by ADS [▶ 164].

5.3.3 CANopen devices

CANopen devices which are not recognized by the TwinCAT System Manager can be incorporated into the network by selecting the box ”CANopen Node”. The CAN(open) messages (PDOs) can be configured directly for these devices. This will guarantee the optimum flexibility of this general CANopen interface.

When using the FC510x / EL6751, this box also enables you to receive and send any CAN identifier - this enables communication with any CAN node. The only condition is the support of at least one of the baud rates [▶ 119] supported by the FC510x / EL6751.

"CAN Node" tab

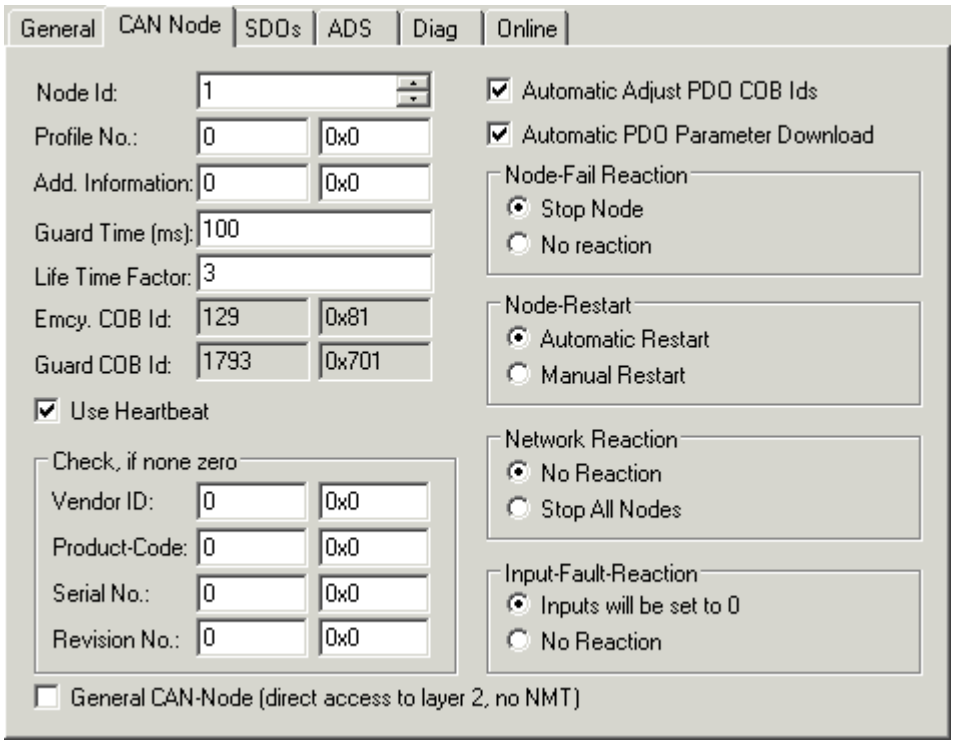


Fig. 106: "CAN Node" tab

Node ID
Enter the general CANopen device node address here. If you select the ”Auto Adapt PDO COB Ids" box, the default identifier for the process data object can also be carried out after changing the node ID.

Profile no.
After CANopen, the parameter 0x1000 "Device Type" contains the number of the supported device profile in both the lowest value bytes. These are entered here and compared at the system StartUp with the device parameters present. If no device profile is supported, the parameter will contain the value 0.

Add info
The additional info is located in both the highest value bytes of the object directory entry 0x1000 (device type).

A set/actual configuration comparison is only made if the profile no. or add. info (i.e. object directory entry 0x1000) is set to a value other than zero. If the expected data at the system start do not comply with the values present, the StartUp of this node will be interrupted and a corresponding error message will appear in the Diag Tab.

Guard time
The guard time determines the interval in which the node is monitored (node guarding). 0 signifies no monitoring. The value entered is rounded up to the next multiple of 10 ms.

Lifetime factor
Guard time x lifetime factor determines the watchdog length for the mutual monitoring of card and CANopen nodes. 0 indicates that the CANopen node is not monitoring the card. At 0 the card takes the guard time directly as the watchdog length.

The FC 510x / EL6751 also support the Heartbeat protocol and initially attempt to start this form of node monitoring on the CANopen node (write access to the objects 0x1016 and 0x1017 in the object dictionary). If this attempt fails, guarding is activated. 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 is transmitted with the smallest configurable guard time (the guard times can be individually set for each node).

Emcy COB Id / Guard COB ID
Identifier for emergency messages or guarding protocol. They result from the node address.

Use Heartbeat
Heartbeat is used for monitoring of the node. If this is deactivated, the guarding is used for monitoring.

Auto-adjust PDO...
Specifies whether TwinCAT should download the PDO communications parameters to the node at the system start.

If the download of the PDOs Parameter Identifier and Transmission Type fails, the card attempts to read these parameters and compare them with the configured values. In this way, it supports only those nodes which, e.g. have implemented the default identifiers as read-only values.

Vendor ID, Product Code, Serial No., Revision No.
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 one part of the value (e.g. vendor ID and product code) - in this case set the not desired parameters to zero.

Node error reaction

- **Stop Node**
  After a recognized node error, the node is set to "Stopped" mode (NMT command "Stop Remote Node"). The node (according to each device profile) can then be switched to a safe mode via the network status machine - SDO addressing is not possible in this mode.
- **no response**
  No NMT stop remote node command after node error

Node Restart

- **Automatic Restart**
  After a recognized node error the card automatically attempts to restart the node. The StartUp attempt is initiated by a node reset command.
- **Manual Restart**
  After a node error, this node remains in error mode and is not restarted automatically. You can actuate a restart via "I/O-Reset" .

Network response

- **no response**
  The failure of a node has no effect on the other bus devices
- **All Nodes Stop**
  After the failure of a node, all other previously started nodes are stopped (NMT stop remote node command). You then need to restart the system.

General CAN Node
If you have selected this checkbox, the entire CANopen network management for this device is deactivated. It is not started, monitored etc. The PDO inputs are detected as pure CAN (2-layer) telegrams and enable the controller to operate in event driven mode.

CANopen terminology

As the CANopen terminology is retained, even in the case of the general CAN nodes, you need to take into account the fact that RxPDOs are the telegrams sent by the FC510x / EL6751 and TxPDOs are the received telegrams.

This option allows any CAN node to be connected to the TwinCAT, if the Baud Rate [▶ 119] and the bit timing parameters comply. The respective protocol can then be simulated within the PLC program. It is also possible to run CANopen devices and general CAN nodes within the same network - if there are no identifier overlaps (the system structure is such that you cannot use an identifier twice).

CANopen PDOs

Process Data Objects [▶ 101] (PDOs) are CAN telegrams which transport process data without a protocol overhead. RxPDOs are received by node, TxPDOs are sent by the node. This description is contained in the System Manager from the perspective of the configured node, i.e. RxPDOs are sent by the TwinCAT, TxPDOs are received by the TwinCAT.

”PDO" tab

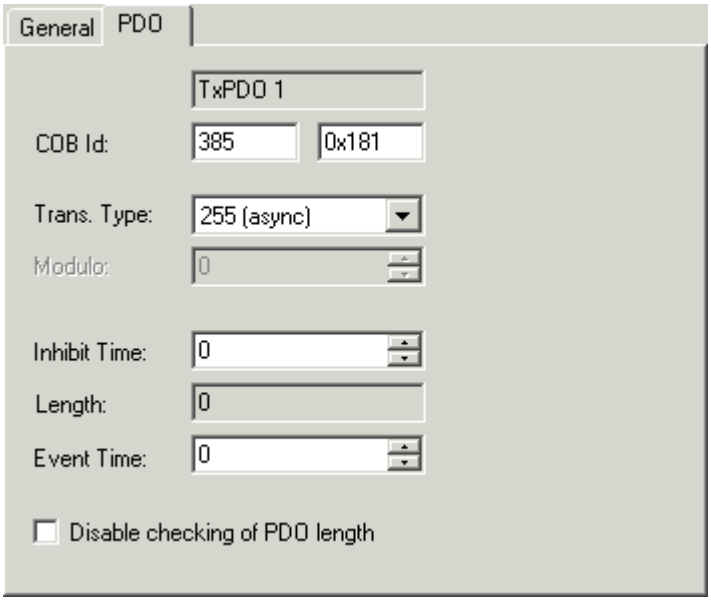


Fig. 107: ”PDO" tab

COB Id
The CAN identifier of this PDO. For every two send and receive PDOs per node, CANopen provides Default Identifiers [▶ 119]. These can then be changed.

Trans.Type
The Transmission Type [▶ 101] determines the send behavior of the PDO. 255 corresponds to the event driven send.

Inhibit Time
Send Delay [▶ 101] between two identical PDOs. Is entered in multiples of 0.1 ms.

Length
The length of the PDO is based on the mapped variables and cannot therefore be edit here.

Event time (FC510x and EL6751 only)
Enter the value for the Event Timer [▶ 101] in ms. For send PDOs (here: RxPDOs, see above) the StartUp of this timer triggers an additional PDO send, for receive PDOs (here: TxPDOs) the arrival of a PDO within the pre-set value is monitored and the box state of the node is changed as appropriate. If 0, the parameter is not transferred to the node.

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. You can view the inputs at the SDO tab. If this behavior is not required, you can deactivate "Auto Download of PDO Parameters" by selecting the checkbox at the CAN node tab.

Deactivate checking of the PDO size
Checkbox for deactivation of the length check of the PDO size.

Tree Representation:

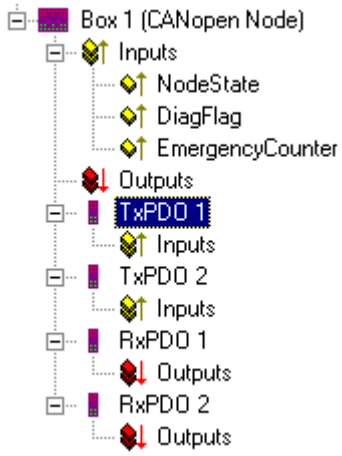


Fig. 108: TwinCAT tree: CANopen Box

TwinCAT firstly provides two send and receive PDOs, with Default Identifiers [▶ 119], for a general CANopen node. Superfluous PDOs can be selected and removed.

TxPDOs are sent by the CANopen node and generally contain inputs. RxPDOs are received by the node, i.e., sent by TwinCAT.

Add variables to the PDOs by right clicking on ”Inputs" and/or ”Outputs" and selecting the corresponding variable(s). If several variables of the same type are inserted with a single action, the offset within the PDO will be created automatically. If variables are inserted one after another, you need to set the corresponding offset (start address within the CAN telegram) for each variable.

Object dictionary entries in TwinCAT

TwinCAT places the PDOs in the displayed order according to the object dictionary entries in the node. This way, for example, the PDO communication parameters of the third listed TxPDO are always written to index 0x1802 – independent of the designation of the PDO in the System Manager. Thus, if only PDO1 and PDO3 are to be used, a PDO2 must also be entered – in this case without assigning variables. PDOs without variables are not transmitted and also not expected.

Context menu:

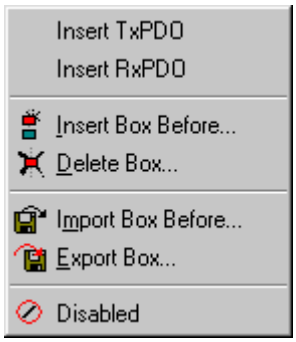


Fig. 109: Context menu for inserting further Tx or Rx-PDOs.

The menu above is obtained by right clicking on the general CANopen node. Here you can insert further Tx PDOs and/or Rx PDOs.

"SDOs" tab

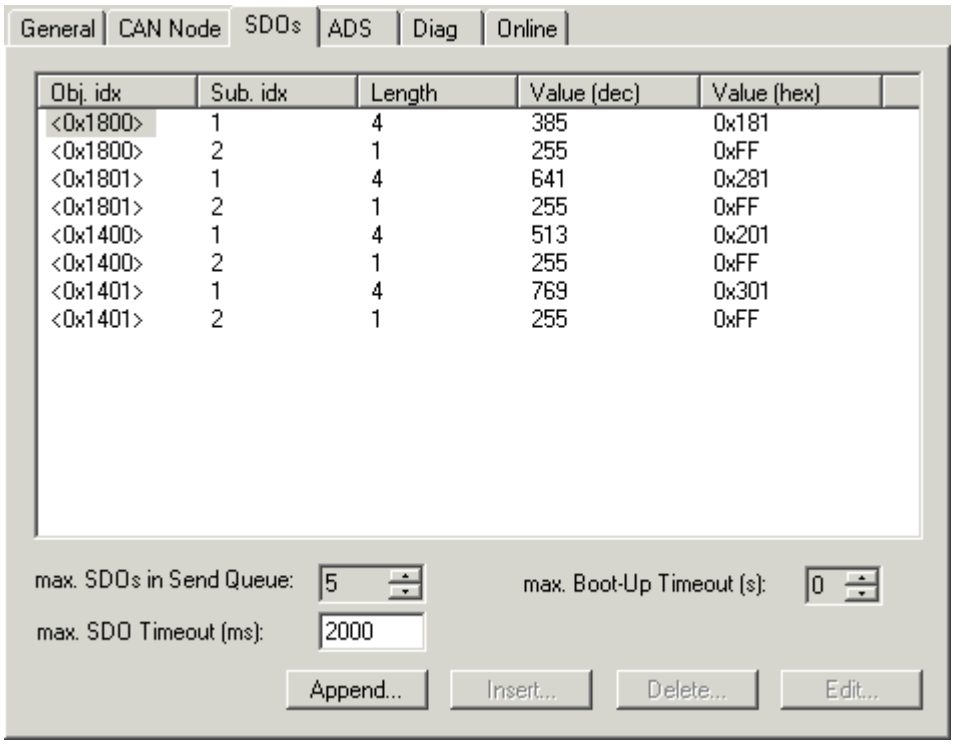


Fig. 110: "SDOs" tab

SDO inputs sent to the node at StartUp are displayed/managed on this page. Inputs with an object index in straight brackets are automatically created on the basis of the updated terminal configuration. Other inputs can be managed using ”Add", ”Insert", ”Delete" and ”Edit".

“ADS” tab

In order to be able to read and write SDO objects during the running time (e.g. from the PLC), the node (Bus Coupler) can be allocated an ADS port (CIFx0-CAN). The FC510x / EL6751 provides an ADS port at all times for every node since the diagnostic information is transported via ADS. These ports can be used to read and write SDO objects using ADS read requests and/or write requests.

The ADS IndexGroup contains the CANopen object index and the ADS IndexOffset contains the CANopen Sub-Index.

5.4 CANopen Communication

5.4.1 Network Management

Simple Boot-Up

CANopen allows the distributed network to boot in a very simple way. After initialization, the modules are automatically in the *Pre-Operational* state. In this state it is already possible to access the object directory using service data objects (SDOs) with default identifiers, so that the modules can be configured. Since default settings exist for all the entries in the object directory, it is in most cases possible to omit any explicit configuration.

Only one CAN message is then required to start the module: Start_Remote_Node: Identifier *0*, two data bytes: 0x01, 0x00. It switches the node into the *Operational* state.

Network Status

The states and the state transitions involved as CANopen boots up can be seen from the state diagram:

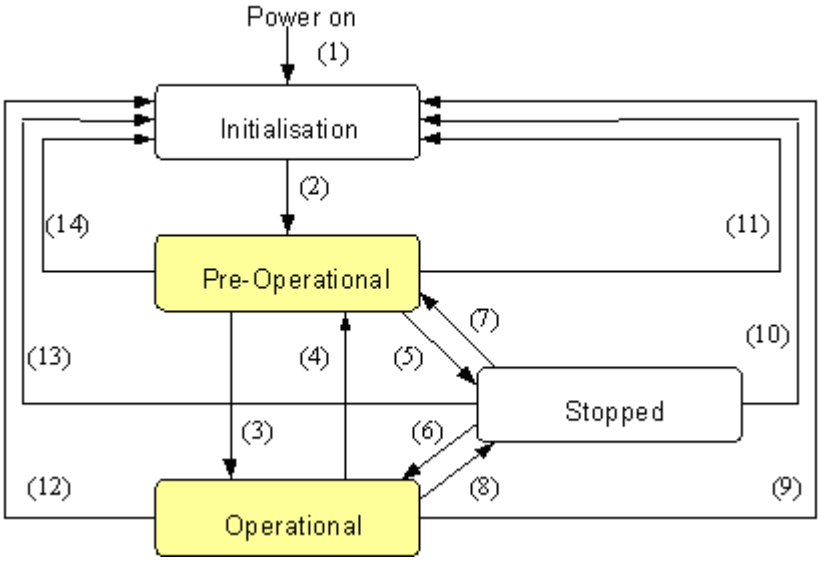


Fig. 111: CANopen bootup state diagram

Pre-Operational

After initialization the Bus Coupler goes automatically (i.e. without the need for any external command) into the *Pre-Operational* state. In this state it can be configured, since the service data objects (SDOs) are already active. The process data objects, on the other hand, are still locked.

Operational

In the *Operational* state the process data objects are also active.

If external influences (such as a CAN error, or absence of output voltage) or internal influences (such as a K-Bus error) mean that it is no longer possible for the Bus Coupler to set outputs, to read inputs or to communicate, it attempts to send an appropriate emergency message, goes into the error state, and thus returns to the *Pre-Operational* state. In this way the NMT status machine in the network master can also immediately detect fatal errors.

Stopped

In the *Stopped* state (formerly: *Prepared*) data communication with the Coupler is no longer possible - only NMT messages are received. The outputs go into the fault state.

State Transitions

The network management messages have a very simple structure: CAN identifier *0*, with two bytes of data content. The first data byte contains what is known as the command specifier (cs), and the second data byte contains the node address, the node address *0* applying to all nodes (broadcast).

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

The following table gives an overview of all the CANopen state transitions and the associated commands (command specifier in the NMT master telegram):

Status 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. Starts the module, enables outputs, starts transmission of PDOs.
(4), (7)	cs = 128 = 0x80	Enter_Pre-Operational. Stops PDO transmission, SDO still active.
(5), (8)	cs = 2 = 0x02	Stop_Remote_Node. Outputs go into the fault state, SDO and PDO switched off.
(9), (10), (11)	cs = 129 = 0x81	Reset_Node. Carries out a reset. All objects are reset to their power-on defaults.
(12), (13), (14)	cs = 130 = 0x82	Reset_Communication. 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 resets node 17:

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

Boot-up message

After the initialization phase and the self-test the Bus Coupler 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 drop), 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 Bus Coupler signals that it can now be configured or started.

Firmware version BA

Up to firmware version BA the emergency identifier was used for the boot up message.

Format of the Boot-up message

11 bit identifier	1 byte of user data							
0x700 (=1792)+ node ID	0x00							

Node Monitoring

Heartbeat and guarding mechanisms are available to monitor failures in the CANopen network. These are of particular importance for CANopen, since modules do not regularly speak in the event-driven mode of operation. 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 non-central peripheral modules, while they themselves can use Life Guarding to detect the failure of the guarding master. Guarding involves the master sending remote frames (remote transmit requests) to the guarding identifier of the slaves that are to be monitored. These reply with the guarding message. This contains the slave's status code and a toggle bit that has to change after every message. If either the status or the toggle bit do not agree with that expected by the NMT master, or if there is no answer at all, the master assumes that there is a slave fault.

Guarding procedure

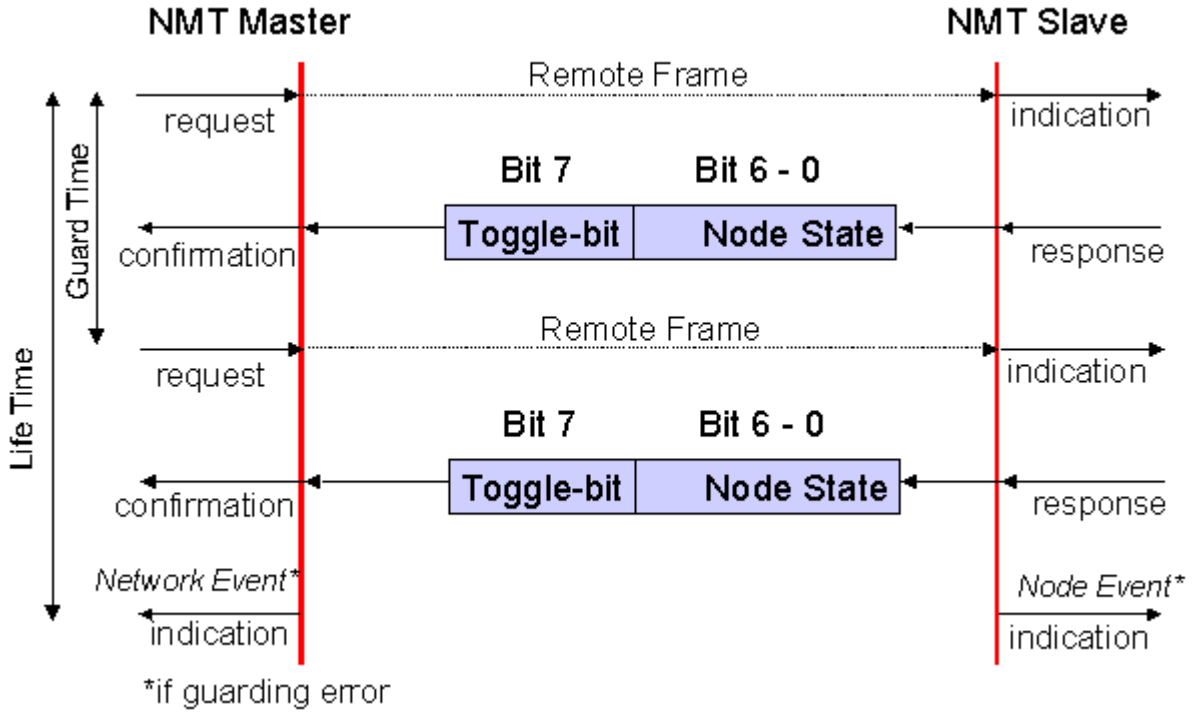


Fig. 112: Schematic diagram: "Guarding procedure"

Protocol

The toggle bit (t) transmitted in the first guarding telegram has the value *0*. After this, the bit must change (toggle) in every guarding telegram so that the loss of a telegram can be detected. The node uses the remaining seven bits to transmit its network status (s):

s	Status
4 = 0x04	Stopped (previously: Prepared)
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), it assumes that the master has failed (the watchdog function). It then puts its outputs into the error state, sends an emergency telegram, and returns to the pre-operational state. After a guarding time-out the procedure can be re-started by transmitting a guarding telegram again.

The node life time is calculated from the guard time (object 0x100C) and life time factor (object 0x100D) parameters:

Life time = guard time x life time factor

If either of these two parameters is "0" (the default setting), the master will not be monitored (no life guarding).

Heartbeat: Node Monitoring without Remote Frame

In the heart beat procedure, each node transmits its status message cyclically on its own initiative. There is therefore no need to use remote frames, and the bus is less heavily loaded than under the guarding procedure.

The master also regularly transmits its heartbeat telegram, so that the slaves are also able to detect failure of the master.

Heartbeat procedure

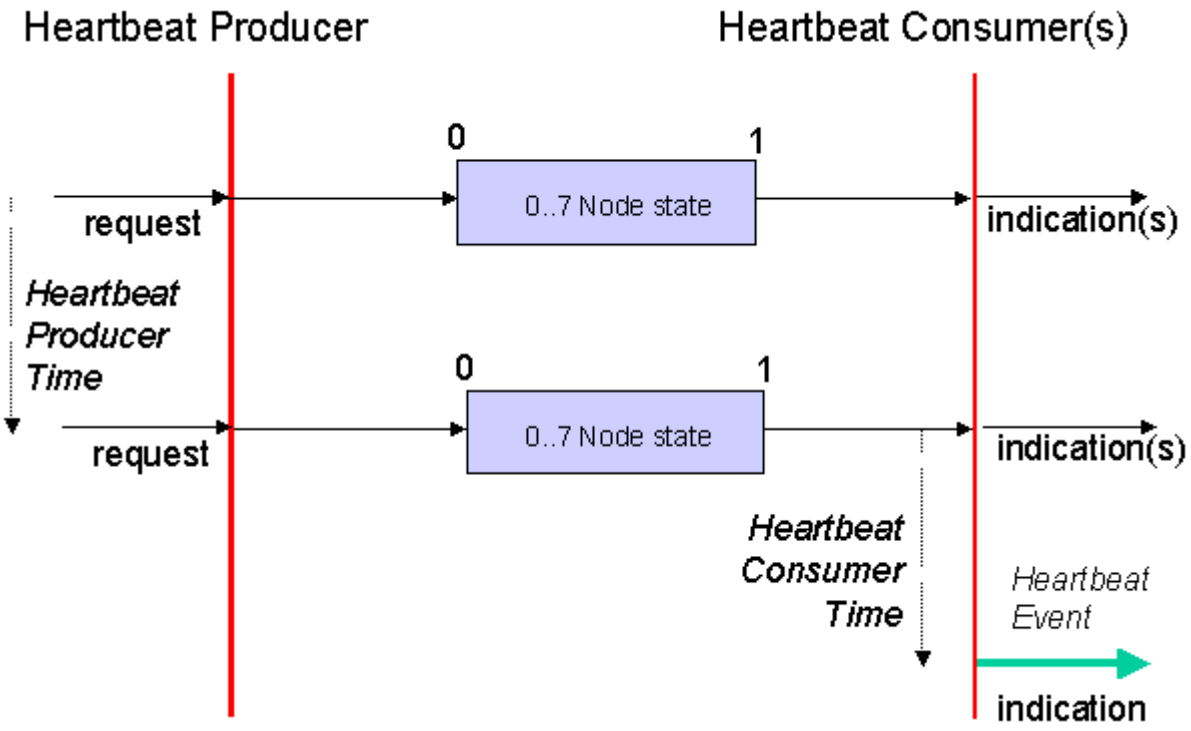


Fig. 113: Schematic diagram: "Heartbeat procedure"

Protocol

The toggle bit is not used in the heart beat procedure. The nodes send their status cyclically (s). See Guarding [▶ 96].

5.4.2 CANopen Master Network management

Automatic CANopen StartUp

After the startup (EL6751: switch-over after SAFEOP) the CANopen master sends a Reset Communication All Nodes command. This is followed by an individual startup for each configured CANopen slave:

SDO	Explanation	Option
Upload Device Type 0x1000	The object 0x1000 of the CANopen slaves is read and compared with the configured value	This SDO is active by default and can be switched off via the configuration (see Advanced button in tab BK51x0 [▶ 86] or CAN node [▶ 88] for the box in the System Manager). If the SDO is active, the startup is aborted if a value other than the configured value is read.
Upload Vendor ID 0x1018:01	The entry 0x1018:01 of the CANopen slave is read and compared with the configured value, if this not equal 0.	This SDO is always active in BK51x0 [▶ 86] Bus Couplers, in general CANopen slaves [▶ 88] only if a vendor ID not equal 0 is entered in the CAN node tab for the box in the System Manager. If the SDO is active, the startup is aborted if a value other than the configured value is read.
Upload Product Code 0x1018:02	The entry 0x1018:02 of the CANopen slave is read and compared with the configured value, if this not equal 0.	This SDO is always active in BK51x0 [▶ 86] Bus Couplers, in general CANopen slaves [▶ 88] only if a product code not equal 0 is entered in the CAN node tab for the box in the System Manager. If the SDO is active, the startup is aborted if a value other than the configured value is read.
Upload Revision Number 0x1018:03	The entry 0x1018:03 of the CANopen slave is read and compared with the configured value, if this not equal 0.	This SDO is never active in BK51x0 [▶ 86] Bus Couplers, in general CANopen slaves [▶ 88] only if a revision number not equal 0 is entered in the CAN node tab for the box in the System Manager. If the SDO is active, the startup is aborted if a value other than the configured value is read.
Upload Serial Number 0x1018:04	The entry 0x1018:04 of the CANopen slave is read and compared with the configured value, if this not equal 0.	This SDO is never active in BK51x0 [▶ 86] Bus Couplers, in general CANopen slaves [▶ 88] only if a serial number not equal 0 is entered in the CAN node tab for the box in the System Manager. If the SDO is active, the startup is aborted if a value other than the configured value is read.
Download SYNC cycle Time 0x1006	The SYNC cycle time is written to object 0x1006 of the CANopen slave, if the SYNC message is sent (the SYNC message is sent if at least one synchronous PDO is configured for any slave).	This SDO is active by default if the SYBC message is sent and can be switched off via the configuration (see Advanced button in tab BK51x0 [▶ 86] or CAN node [▶ 88] for the box in the System Manager). If the SDO is active, the startup is aborted if an SDO abort has occurred.
Download PDO ID (0x1400+y:01 or 0x1800+y:01)	The COB-ID is written to for each configured PDO.	These SDOs are active by default for general CANopen slaves [▶ 88] and can be switched off via the CAN node [▶ 88] tab. For Bus Couplers the PDOs are configured via object 0x5500. Therefore these SDOs are not active for BK51x0 [▶ 86] Bus Couplers.

SDO	Explanation	Option
Upload PDO ID (0x1400+y:01 or 0x1800+y:01)	If an SDO abort occurred during the PDO COB-ID download, the system tries to read the entry.	This SDO is only active if a fault occurred in the download of the respective PDO COB ID. If the SDO is active, the startup is aborted if a value other than the configured value is read.
Download PDO Transmission Type (0x1400+y:02 or 0x1800+y:02)	The transmission type is described for each configured PDO	These SDOs are active by default for general CANopen slaves [▶ 88] and can be switched off via the CAN node [▶ 88] tab. For Bus Couplers the transmission type is only distinguished for digital (PDO 1) and analog (PDO 2) terminals, if object 0x5500 is written to on startup. Therefore, for BK51x0 [▶ 86] Bus Couplers these SDOs are only active for PDOs 1 and 2.
Upload PDO Transmission Type (0x1400+y:02 or 0x1800+y:02)	If an SDO abort occurred during the PDO transmission type download, the system tries to read the entry.	This SDO is only active if a fault occurred in the download of the respective transmission type PDO. If the SDO is active, the startup is aborted if a value other than the configured value is read.
Download PDO Inhibit Time (0x1400+y:03 or 0x1800+y:03)	The inhibit time is written to for each configured PDO.	These SDOs are active for general CANopen slaves [▶ 88], if an inhibit time greater than 0 is configured on the PDO [▶ 88] tab of the respective PDO. For Bus Couplers there is only one inhibit time for all PDOs, if the PDOs are configured via the object 0x5500. The SDOs are active if this inhibit time on tab BK51x0 [▶ 86] is greater than 0.
Upload PDO Inhibit Time (0x1400+y:03 or 0x1800+y:03)	If an SDO abort occurred during the PDO inhibit time download, the system tries to read the entry.	This SDO is only active if a fault occurred in the download of the respective PDO inhibit time. If the SDO is active, the startup is aborted if a value other than the configured value is read.
Download PDO Event Time (0x1400+y:05 or 0x1800+y:05)	The event time is written to for each configured PDO.	These SDOs are active for general CANopen slaves [▶ 88], if an event time greater than 0 is configured on the PDO [▶ 88] tab of the respective PDO. For Bus Couplers there is only one event time for all PDOs, if the PDOs are configured via the object 0x5500. The SDOs are active if this event time on tab "BK51x0 [▶ 86]" is greater than 0.
Upload PDO Event Time (0x1400+y:05 or 0x1800+y:05)	If an SDO abort occurred during the PDO event time download, the system tries to read the entry.	This SDO is only active if a fault occurred in the download of the respective PDO event time. If the SDO is active, the startup is aborted if a value other than the configured value is read.
Download Producer Heartbeat 0x1017	The guard time is written to object 0x1017 of the CANopen slave.	This SDO is active if the guard time and the life time factor on the tabs BK51x0 [▶ 86] or CAN node [▶ 88] are not equal 0. If the SDO is active, the startup is aborted if an SDO timeout has occurred.

SDO	Explanation	Option
Download Consumer Heartbeat 0x1016:01	The object 0x1016:01 of the CANopen slaves is multiplied with the guard time and described with the life time factor	This SDO is active if the guard time and the life time factor on the tabs BK51x0 [▶ 86] and CAN node [▶ 88] are not equal 0 and no abort occurred during the download of the producer heartbeat. If the SDO is active, the startup is aborted if an SDO abort has occurred.
Download Guard Time 0x100C	The guard time is written to object 0x100C of the CANopen slave.	This SDO is active if the guard time and the life time factor on the tabs BK51x0 [▶ 86] and CAN node [▶ 88] are not equal 0 and an SDO abort (no SDO timeout) occurred during the download of the producer heartbeat. If the SDO is active, the startup is aborted if an SDO abort has occurred.
Download Life Time Factor 0x100D	The life time factor is written to object 0x100D of the CANopen slave.	This SDO is active if the guard time and the life time factor on the tabs BK51x0 [▶ 86] and CAN node [▶ 88] are not equal 0 and an SDO abort (no SDO timeout) occurred during the download of the producer heartbeat. If the SDO is active, the startup is aborted if an SDO abort has occurred.
Download further startup SDOs	Further startup SDOs are written	All further startup SDOs are written that are listed on the SDOs tab for BK51x0 [▶ 86] Bus Couplers or general CANopen slaves [▶ 88].
Start Node	The CANopen slave is started	The CANopen slave startup is active by default and can be switched off via the configuration (see Advanced button in tab BK51x0 [▶ 86] or CAN node [▶ 88] for the box in the System Manager). If the CANopen slaves startup is not active, it can be started manually [▶ 101].
Start All Nodes	All CANopen slaves are started	Once all CANopen slaves have been started individually, a start command is sent to all CANopen slaves, if the automatic start was not deactivated in a CANopen slave.
Waiting for TxPDOs		The NodeState is set to 23 as long as not all configured TxPDOs of the CANopen slave were received. If the SDOs tab for BK51x0 [▶ 86] Bus Couplers or general CANopen slaves [▶ 88] is set to restart the CANopen slave if 10 s after the startup no configured TxPDO was received (not active by default), the complete startup is repeated if this monitoring function is triggered.
Sending the RxPDOs		The configured RxPDOs are sent to the CANopen slave 1 second after the CANopen slave was started.

SDO	Explanation	Option
Monitoring the synchronous TxPDOs		Monitoring of the synchronous TxPDOs commences when they were received for the first time. If the transmission type is set to 1, this TxPDO must be received in the SYNC cycle, otherwise the node state switches to 40 or 22 and the CANopen slave is treated according to configured error response. The time slot ends after the input shift time has elapsed (EL6751, with a SYNC multiplier greater than 1 the input shift time in the last EtherCAT cycle counts before the next SYNC cycle commences) or once all synchronous RxPDOs were sent (FC51xx, CX1500-M510). The monitoring can be made less strict by setting an event time not equal 0 in the corresponding TxPDO. In this case the CANopen master is tolerant for one SYNC cycle, i.e. the node state is not set to 22 until the TxPDO has failed twice in succession. For transmission types greater than 1 the CANopen master is also tolerant for one cycle before a fault is detected and the node state is set to 22.
Monitoring the asynchronous TxPDOs		Monitoring of the asynchronous TxPDOs is only active if their event time is configured with greater than 0 and commences when they are received for the first time. If the TxPDO is not received within twice the event time, the node state is set to 22 and the CANopen slave is treated according to the configured error response.
Error response		If a fault is detected in a CANopen slave during TxPDO monitoring or during guarding/heartbeat, the node state switches to a value not equal 0 and configured error response is triggered (according to tab BK51x0 [▶ 86] or CAN node [▶ 88]). In the default setting the CANopen slave is stopped and then restarted (with communication reset).

Manual network management

The CANopen state (STOPPED, PRE-OPERATIONAL, OPERATIONAL) of a CANopen slave can be changed via ADS write control. In this case the AMS address should be set as for SDO communication. The other parameter are listed in the following table:

ADS State	Device State	CANopen state transition
ADSSTATE_RUN (5)	0	OP->PREOP
ADSSTATE_RUN (5)	1	PREOP->OP
ADSSTATE_STOP (6)	0	OP->STOP
ADSSTATE_RUN (5)	1	STOP->OP (with communication reset)
ADSSTATE_RUN (5)	3	STOP->OP (without communication reset)
ADSSTATE_STOP (6)	0	PREOP->STOP
ADSSTATE_RUN (5)	2	STOP->PREOP (without communication reset)

5.4.3 Process Data Objects (PDO)

Introduction

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 the multi-master bus access protocol allows CAN to offer other methods. Under CANopen the process data is not transferred in a master/slave procedure, but follows instead the producer-consumer model. In this model, a bus node transmits its

data, as a producer, on its own accord. This might, for example, be triggered by an event. All the other nodes listen, and use the identifier to decide whether they are interested in this telegram, and handle it accordingly. These are the consumers.

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 PDOs each correspond to a CAN telegram, whose specific CAN identifier is used to allocate them and to determine their priority. Receive PDOs (RxPDOs) and transmit PDOs (TxPDOs) are distinguished, the name being chosen from the point of view of the device: an input/output module sends its input data with TxPDOs and receives its output data in the RxPDOs. **This naming convention is retained in the TwinCAT System Manager.**

Communication parameters

The PDOs can be given different communication parameters according to the requirements of the application. Like all the CANopen parameters, these are also available in the device's object directory, and can be accessed by means of the service data objects. The parameters for the receive PDOs are at index 0x1400 (RxPDO1) onwards. There can be up to 512 RxPDOs (ranging up to index 0x15FF). In the same way, the entries for the transmit PDOs are located from index 0x1800 (TxPDO1) to 0x19FF (TxPDO512).

The Beckhoff Bus Couplers or Fieldbus Coupler Box modules make 16 RxPDO and TxPDOs available for the exchange of process data (although the figure for Economy and LowCost BK5110 and LC5100 Couplers and the Fieldbus Boxes is 5 PDOs each, since these devices manage a lower quantity of process data). The FC510x CANopen master card supports up to 192 transmit and 192 receive PDOs for each channel - although this is restricted by the size of the DPRAM. The EL6751 CANopen terminal dynamically organizes the process image; i.e. the process data are written in succession, enabling a higher data transmission rate. Up to 32 TxPDOs and 32 RxPDOs can be handled in slave mode.

For each existing process data object there is an associated communication parameter object. The TwinCAT System Manager 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.

PDO Identifier

The most important communication parameter in a PDO is the CAN identifier (also known as the communication object identifier, or COB-ID). It is used to identify the data, and determines their priority for bus access. For each CAN data telegram there may only be one sender node (producer), although all messages sent in the CAN broadcast procedure can be received, as described, by any number of nodes (consumers). Thus a node can make its input information available to a number of bus devices at the same time - even without transferring them through a logical bus master. The identifier is located in sub-index 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 identifier itself. The data width of the object of 32 bits also allows 29-bit identifiers in accordance with CAN 2.0B to be entered, although the default identifiers always refer to the more usual 11-bit versions. Generally speaking, CANopen is economical it its use of the available identifiers, so that the use of the 29-bit versions remains limited to unusual applications. It is therefore also not supported by a Beckhoff's CANopen devices. The highest bit (bit 31) can be used to activate the process data object or to turn it off.

A complete identifier list [▶ 188] is provided in the appendix.

PDO linking

In the system of default identifiers, all the nodes (here: slaves) communicate with one central station (the master), since slave nodes do not listen by default to the transmit identifier of any other slave node.

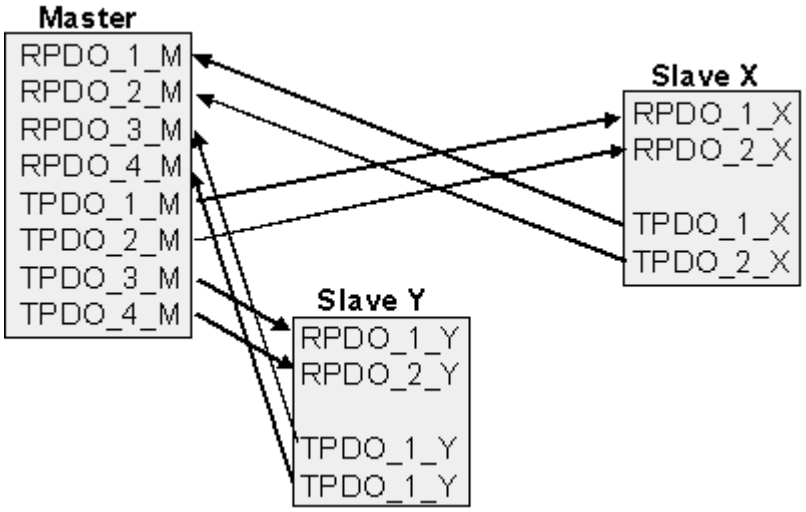


Fig. 114: Default identifier allocation: Master/Slave

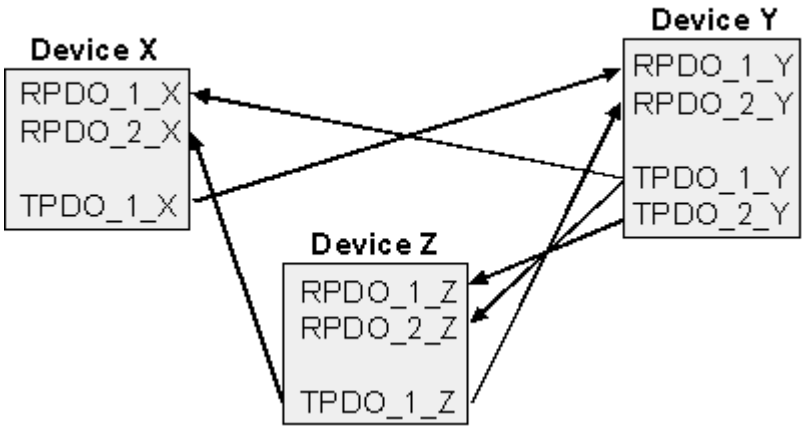


Fig. 115: PDO linking: Peer to Peer

If the consumer-producer model of CANopen PDOs is to be used for direct data exchange between nodes (without a master), the identifier allocation must be appropriately adapted, so that the TxPDO identifier of the producer agrees with 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.

PDO Communication Types: Overview

CANopen offers a number of possible ways to transmit process data (see also: Notes on PDO Parameterization [▶ 109]).

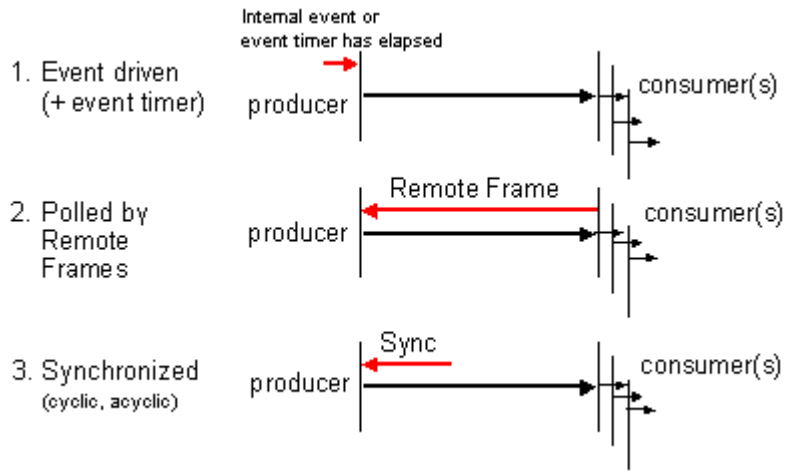


Fig. 116: Diagram: CAN process data transmission

Event driven

The ”event" is the alteration of an input value, the data being transmitted immediately after this change. The event-driven flow can make optimal use of the bus bandwidth, since instead of the whole process image it is only the changes in it that are transmitted. A short reaction time is achieved at the same time, since when an input value changes it is not necessary to wait for the next interrogation from a master.

As from CANopen Version 4 it is possible to combine the event driven type of communication with a cyclic update. Even if an event has not just occurred, event driven TxPDOs are sent after the event timer has elapsed. If an event does occur, the event timer is reset. For RxPDOs the event timer is used as a watchdog in order to monitor the arrival of event driven PDOs . If a PDO does not arrive within a set period of time, the bus node adopts the error state.

Polled

The PDOs can also be polled by data request telegrams (remote frames). In this way it is possible to get the input process image of event-driven inputs onto the bus, even when they do not change, for instance through a monitoring or diagnostic device brought into the network while it is running. The time behavior of remote frame and response telegrams depends on what CAN controller is in use. Components with full integrated message filtering ("FullCAN") usually answer a data request telegram immediately, transmitting data that is waiting in the appropriate transmit buffer - it is the responsibility of the application to see that the data there is 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 can 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 nodes as a trigger for reading the inputs or for setting the outputs.

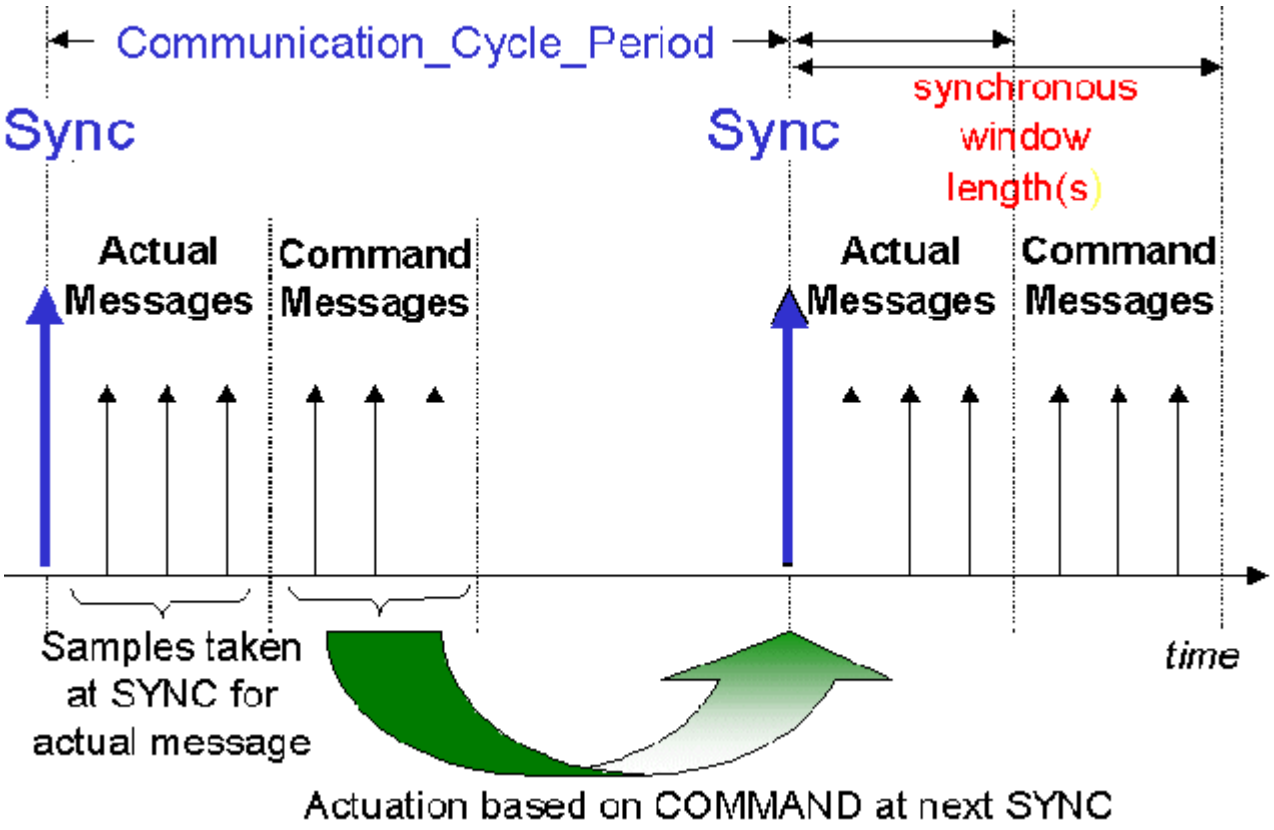


Fig. 117: Diagram: CAN "SYNC" telegram

PDO transmission types: Parameterization

The PDO transmission type parameter specifies how the transmission of the PDO is triggered, or how received PDOs are handled.

Transmission type	Cyclical	Acyclical	Synchronous	Asynchronous	Only RTR
0		X	X		
1-240	X		X		
241-251	- reserved -				
252			X		X
253				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.

Acyclic Synchronous

PDOs of transmission type 0 function synchronously, but not cyclically. An RxPDO is only evaluated after the next SYNC telegram has been received. 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. A device whose TxPDO is configured for transmission type 0 acquires its input data when it receives the SYNC (synchronous process image) and then transmits it if the data correspond to an event (such as a change in input) having occurred. Transmission type 0 thus combines transmission for reasons that are event driven with a time for transmission (and, as far as possible, sampling) and processing given by the reception of "SYNC".

Cyclic Synchronous

In transmission types 1-240 the PDO is transmitted cyclically: after every ”nth" SYNC (n = 1...240). Since transmission types can be combined on a device as well as in the network, it is possible, for example, for a fast cycle to be agreed for digital inputs (n = 1), whereas the data for analog inputs is transmitted in a slower cycle (e.g. n = 10). RxPDOs do not generally distinguish between transmission types 0...240: a PDO that has been received is set to valid when the next SYNC is received. 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 error state.

The FC510x card / EL6751 terminal fully support the synchronous communication method: transmitting the SYNC telegram is coupled to the linked task, so that new input data is available every time the task begins. If a synchronous PDO does not arrive, this is detected and reported to the application.

Only RTR

Transmission types 252 and 253 apply to process data objects that are transmitted exclusively on request by a remote frame. 252 is synchronous: when the SYNC is received the process data is acquired. It is only transmitted on request. 253 is asynchronous. The data here is acquired continuously, and transmitted on request. This type of transmission is not generally recommended, because fetching input data from some CAN controllers is only partially supported. Because, furthermore, the CAN controllers sometimes answer remote frames automatically (without first requesting up-to-date input data), there are circumstances in which it is questionable whether the polled data is up-to-date. Transmission types 252 and 253 are for this reason not supported by the Beckhoff PC cards / terminals.

Asynchronous

The transmission types 254 + 255 are asynchronous, but may also be 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.

Inhibit time

The ”inhibit time" parameter can be used to implement a ”transmit filter" that does not increase the reaction time for relatively new input alterations, but is active for changes that follow immediately afterwards. The inhibit time (transmit delay time) specifies the minimum length of time that must be allowed to elapse between the transmission of two of the same telegrams. If the inhibit time is used, the maximum bus loading can be determined, so that the worst case latency can then be found.

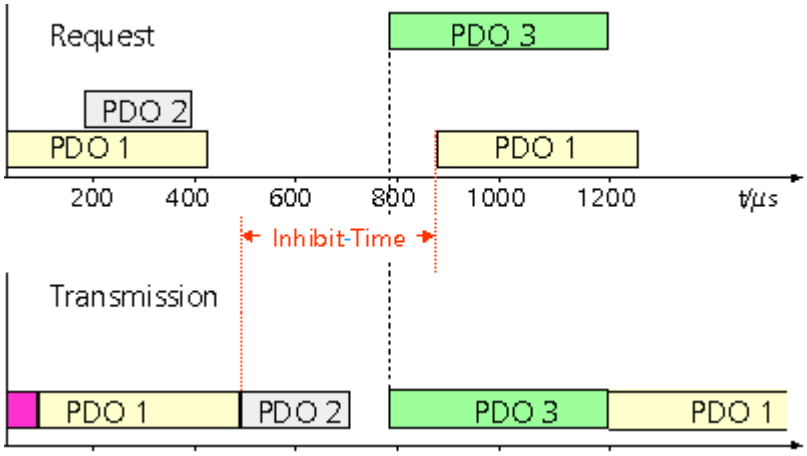


Fig. 118: Timing diagram: "Inhibit time"

Although the Beckhoff FC510x PC cards / EL6751 terminal can parameterize the inhibit time on slave devices, they do not themselves support it. The transmitted PDOs become automatically spread out (transmit delay) as a result of the selected PLC cycle time - and there is little value in having the PLC run faster than the bus bandwidth permits. The bus loading, furthermore, can be significantly affected by the synchronous communication.

Event Timer

An event timer for transmit PDOs can be specified by sub-index 5 in the communication parameters. Expiry of this timer is treated as an additional event for the corresponding PDO, so that the PDO will then be transmitted. If the application event occurs during a timer period, it will also be transmitted, and the timer is reset.

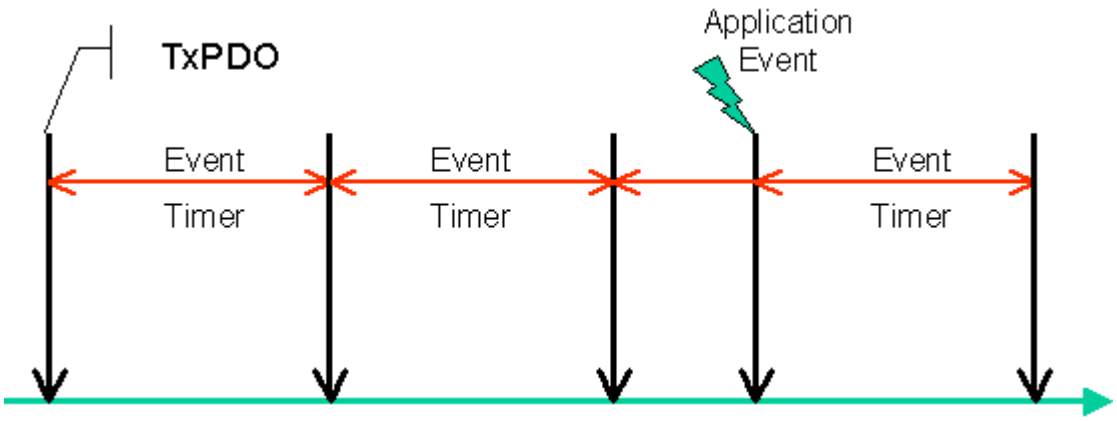


Fig. 119: Time representation of the event timer

In the case of receive PDOs, the timer is used to set a watchdog interval for the PDO: the application is informed if no corresponding PDO has been received within the set period. The FC510x / EL6751 can in this way monitor each individual PDO.

Notes on PDO Parameterization [▶ 109]

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.

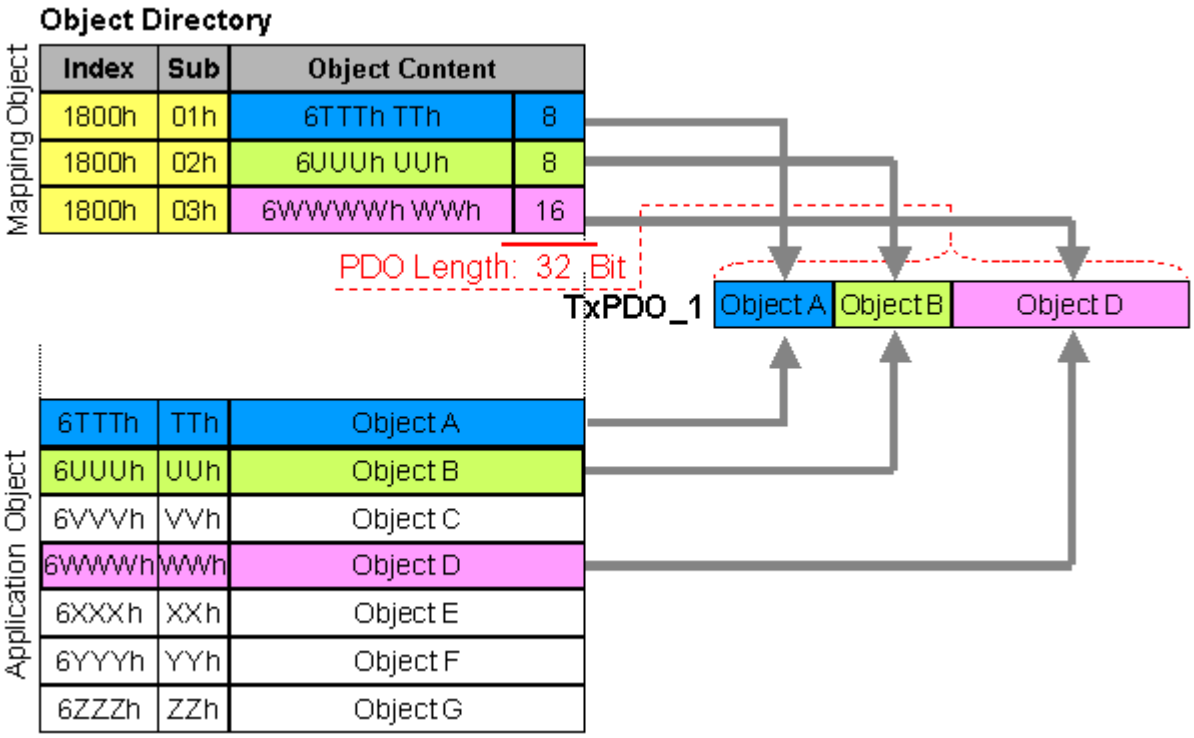


Fig. 120: Mapping representation

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.

5.4.4 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).

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 BECKHOFF FC510x PC cards / EL6751 CANopen terminal are capable of synchronizing the CANopen bus system with the cycles of the application program (PLC or NC).

  The guaranteed reaction time under cyclic synchronous communication is always at least as long as the cycle time, and 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 [▶ 101], 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 [▶ 101] 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 [▶ 101].

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 sample 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 [▶ 101]). In this sample 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 cannot 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 FC510x CANopen master cards / EL6751 CANopen master terminal display the bus load 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. [▶ 107]

5.4.5 Service Data Objects (SDO)

The parameters listed in the object directory are read and written by means of service data objects. These SDOs are *Multiplexed Domains*, i.e. data structures of any size that have a multiplexer (address). The multiplexer consists of a 16-bit index and an 8-bit sub-index that address the corresponding entries in the object directory.

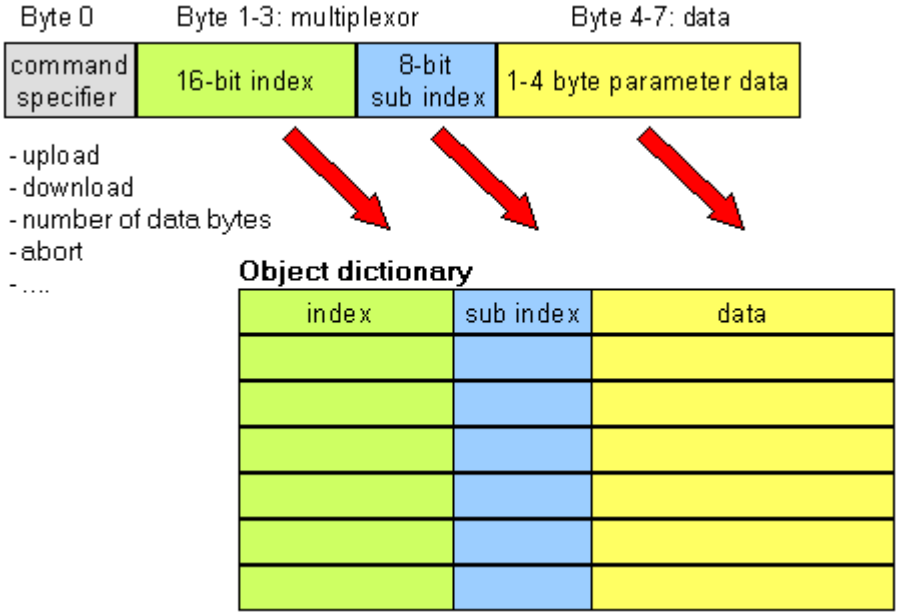


Fig. 121: SDO protocol: access to the object directory

The CANopen Bus Couplers are servers for the SDO, which means that at the request of a client (e.g. of the IPC or the PLC) they make data available (upload), or they receive data from the client (download). This involves a handshake between the client and the server.

When the size of the parameter to be transferred is not more than 4 bytes, a single handshake is sufficient (one telegram pair): For a download, the client sends the data together with its index and sub-index, and the server confirms reception. For an upload, the client requests the data by transmitting the index and sub-index of the desired parameter, and the server sends the parameter (including index and sub-index) in its answer telegram.

The same pair of identifiers is used for both upload and download. The telegrams, which are always 8 bytes long, encode the various services in the first data byte. All parameters with the exception of objects 1008h, 1009h and 100Ah (device name, hardware and software versions) are only at most 4 bytes long, so this description is restricted to transmission in expedited transfer.

Protocol

The structure of the SDO telegrams is described below.

Client -> Server, Upload Request

11 bit identifier	8 byte user data							
0x600 (=1536dec) + node ID	0x40	Index0	Index1	SubIdx	0x00	0x00	0x00	0x00

Parameter	Explanation
Index0	Index low byte (Unsigned16, LSB)
Index1	Index high byte (Unsigned16, MSB)
SubIdx	Sub-index (Unsigned8)

Client -> Server, Upload Response

11 bit identifier	8 byte user data							
0x580 (=1408dec) + node ID	0x4x	Index0	Index1	SubIdx	Data0	Data1	Data2	Data3

Parameter	Explanation
Index0	Index low byte (Unsigned16, LSB)
Index1	Index high byte (Unsigned16, MSB)
SubIdx	Sub-index (Unsigned8)
Data0	Data low low byte (LLSB)
Data3	Data high high byte (MMSB)

Parameters whose data type is Unsigned8 are transmitted in byte D0, parameters whose type is Unsigned16 use D0 and D1.

The number of valid data bytes is coded as follows in the first CAN data byte (0x4x):

Number of parameter bytes	1	2	3	4
First CAN data byte	0x4F	0x4B	0x47	0x43

Client -> Server, Download Request

11 bit identifier	8 byte user data							
0x600 (=1536dec) + node ID	0x22	Index0	Index1	SubIdx	Data0	Data1	Data2	Data3

Parameter	Explanation
Index0	Index low byte (Unsigned16, LSB)
Index1	Index high byte (Unsigned16, MSB)
SubIdx	Sub-index (Unsigned8)
Data0	Data low low byte (LLSB)
Data3	Data high high byte (MMSB)

It is optionally possible to give the number of valid parameter data bytes in the first CAN data byte

Number of parameter bytes	1	2	3	4
First CAN data byte	0x2F	0x2B	0x27	0x23

This is, however, not generally necessary, since only the less significant data bytes up to the length of the object directory entry that is to be written are evaluated. A download of data up to 4 bytes in length can therefore always be achieved in BECKHOFF bus nodes with 22 h in the first CAN data byte.

Client -> Server, Download Response

11 bit identifier	8 byte user data							
0x580 (=1408dec) + node ID	0x60	Index0	Index1	SubIdx	0x00	0x00	0x00	0x00

Parameter	Explanation
Index0	Index low byte (Unsigned16, LSB)
Index1	Index high byte (Unsigned16, MSB)
SubIdx	Sub-index (Unsigned8)

Breakdown of Parameter Communication

Parameter communication is interrupted if it is faulty. The client or server send an SDO telegram with the following structure for this purpose:

11 bit identifier	8 byte user data							
0x580 (client) or 0x600(server) + node ID	0x80	Index0	Index1	SubIdx	Error0	Error1	Error2	Error3

Parameter	Explanation
Index0	Index low byte (Unsigned16, LSB)
Index1	Index high byte (Unsigned16, MSB)
SubIdx	Sub-index (Unsigned8)
Error0	SDO error code low low byte (LLSB)
Error3	SDO error code high high byte (MMSB)

List of SDO error codes (reason for abortion of the SDO transfer):

SDO error code	Explanation
0x05 03 00 00	Toggle bit not changed
0x05 04 00 01	SDO command specifier invalid or unknown
0x06 01 00 00	Access to this object is not supported
0x06 01 00 02	Attempt to write to a Read_Only parameter
0x06 02 00 00	The object is not found in the object directory
0x06 04 00 41	The object cannot be mapped into the PDO
0x06 04 00 42	The number and/or length of mapped objects would exceed the PDO length
0x06 04 00 43	General parameter incompatibility
0x06 04 00 47	General internal error in device
0x06 06 00 00	Access interrupted due to hardware error
0x06 07 00 10	Data type or parameter length do not agree or are unknown
0x06 07 00 12	Data type does not agree, parameter length too great
0x06 07 00 13	Data type does not agree, parameter length too short
0x06 09 00 11	Sub-index not present
0x06 09 00 30	General value range error
0x06 09 00 31	Value range error: parameter value too great
0x06 09 00 32	Value range error: parameter value too small
0x06 0A 00 23	Resource not available
0x08 00 00 21	Access not possible due to local application
0x08 00 00 22	Access not possible due to current device status

Further, manufacturer-specific error codes have been introduced for register communication (index 0x4500, 0x4501):

SDO error code	Explanation
0x06 02 00 11	Invalid table: Table or channel not present
0x06 02 00 10	Invalid register: table not present
0x06 01 00 22	Write protection still set
0x06 07 00 43	Incorrect number of function arguments
0x06 01 00 21	Function still active, try again later
0x05 04 00 40	General routing error
0x06 06 00 21	Error accessing BC table
0x06 09 00 10	General error communicating with terminal
0x05 04 00 47	Time-out communicating with terminal

5.4.6 EL6751- SDO communication

CANopen SDO (Service Data Object) communication is used to read or write any parameters in the CANopen bus node's object directory. The EL6751 CANopen terminal uses the SDO communication for the configuration of the communication parameters when starting up. Two types of application-specific SDO communication are additionally possible:

1. Downloading Application-Specific Parameters when Starting Up

The appropriate parameters are to be entered here in the System Manager for the corresponding node in the "SDO" tab. The objects that result from the configuration in the CAN node tab appear in square brackets. Any desired number of object directory entries can then be inserted.

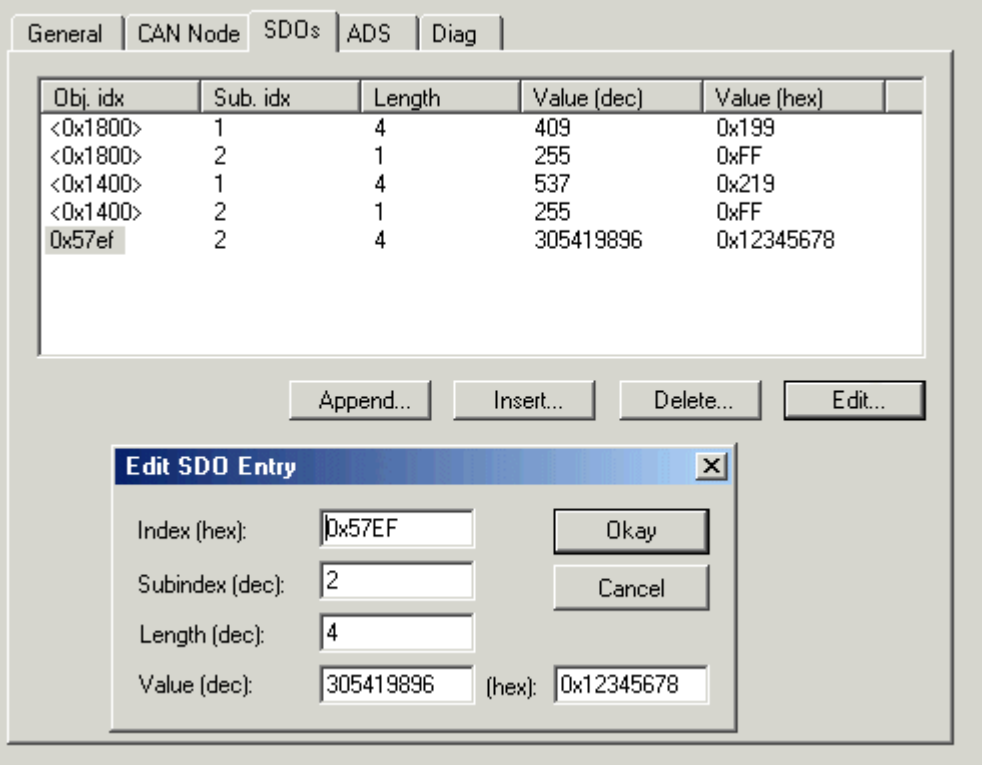


Fig. 122: SDO tab, editing an SDO entry

The terminal expects a positive acknowledgement of the parameter download from the relevant bus device. If it was not possible to write a parameter (the bus device has aborted the SDO) the terminal then attempts to read the corresponding value back and to compare it with the value that was to be written. This is because it could, for instance, be a read-only value, and therefore already correctly configured within the bus device. If the values match the terminal goes to the next parameter entry.

2. Upload and Download at Runtime via ADS

It is possible to perform SDO accesses to the bus devices' object directories using Beckhoff's ADS communication when the system is running. This is also possible from the PLC, from the NC, from the OPC server, from ActiveX controls or from any other ADS device.

The whole SDO protocol is handled by the terminal. Using the ADS Write or ADS Read functions the parameters are transferred to the terminal, and the data is transferred (write) or fetched (read). The "IDXGRP" parameter here corresponds to the 16-bit index in the CANopen object directory, while "IDXOFFS" corresponds to the 8 bit sub-index in the CANopen object directory. Details about the ADS function blocks can be found in the TwinCAT documentation (Beckhoff Information System).

The ADS function block parameters are represented as follows in the SDO parameters:

ADSREAD / ADSWRITE

Parameter	Description
NETID	The NetID is a string, 23 bytes in length, and is formed by default from the IP address of the computer with an additional two bytes. It addresses the EL6751 and can be taken from the "ADS" tab in the System Manager.
PORT	Contains the ADS device's port number - this is the port number of the CANopen bus device that is to be addressed.
IDXGRP	**Corresponds to the 16 bit index in the CANopen object directory.**
IDXOFFS	**Corresponds to the 8 bit sub-index in the CANopen object directory.**
LEN	The length of the parameter that is to be read or written, in bytes.
DESTADDR (only ADSREAD)	Contains the address of the buffer which is to receive the data that has been read. The programmer is himself responsible for dimensioning the buffer to a size that can accept 'LEN' bytes. The buffer can be a single variable, an array or a structure, whose address can be found with the ADR operator.
SRCADDR (only ADSWRITE)	Contains the address of the buffer from which the data to be written is to be fetched. The programmer is himself responsible for dimensioning the buffer to such a size that 'LEN' bytes can be taken from it. The buffer can be a single variable, an array or a structure, whose address can be found with the ADR operator.
READ	The ADS command is triggered by a rising edge at this input.
TIMEOUT	States the time before the function is cancelled.
BUSY	This output remains TRUE until the function block has executed a command, but at the longest for the duration supplied to the 'Timeout' input. While Busy = TRUE, no new command will be accepted at the inputs. Please note that it is not the execution of the service but its acceptance whose time is monitored.
ERR	This output is switched to TRUE if an error occurs during the execution of the command.
ERRID	Contains the command-specific error code of the most recently executed command. Is reset to 0 by the execution of a command at the inputs.

The ERRID is a 32-bit value. The Low word (bits 0 to 15) contains the general ADS ERROR CODES, while the High word (bits 16 to 31) returns SDO-specific error codes:

MSB				LSB
Bit 31	Bit 30...24	Bit 23...20	Bit 19..16	Bit 15...0
1	Bits 6 to 0 of the SDO error code	Bits 19 to 16 of the SDO error code	Bits 27 to 24 of the SDO error code	ADS ERROR code, see chapter Error handling and diagnostics [▶ 168] for meaning

If one of the values SDO Additional Code, SDO Error Code or SDO Error Class is larger than the available data width (hidden bits set), then the value 0x2115 is returned in the High word (bits 16 to 31).

Sample: SDO Read via ADS

In the following sample program (structured text) for the use of ADS services for SDO communication, object 0x1000, sub-index 0, from the node with port number 0x1001 is read. The DeviceType is CANopen. This is coded as UnSigned32, and is therefore 4 bytes long.

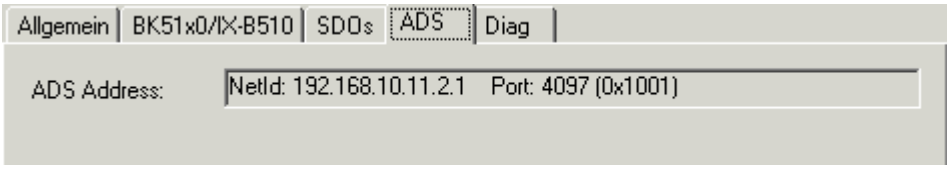


Fig. 123: ADS tab

```
SDO_READ(

StartReading := ReadStart,
CO_Index := 16#1000,
CO_SubIndex := 16#0,
DataLength := 4,
PortNr := 16#1001,
ADSNetID:='192.168.10.11.2.1'
);

IF SDO_READ.ReadDataAvailable THEN
ReadStart := FALSE;
ReadError := SDO_READ.Error;
ReadData := SDO_READ.ReadData;
END_IF
```

The SDO_READ function block that has been called in turn calls the ADSREAD function a number of times. It looks like this (starting with the variable declaration):

```
FUNCTION_BLOCK SDO_READ
VAR_INPUT

ADSNetID:STRING(23); (* The AMSNetID addresses the EL6751. Can be empty if only one local single cha
nnel card is present*)

PortNr:WORD;         (* This Port No. addresses the CANopen Node (see System Manager) *)

CO_Index:DWORD;      (* This is the Index of the CANopen Object Dictionary Entry*)

CO_SubIndex:DWORD;   (* This is the Sub-Index of the CANopen Object Dictionary Entry*)

DataLength:DWORD;    (* This is the Length of the CANopen Object Dictionary Entry*)

StartReading:BOOL;   (* only reset to FALSE after ReadDataAvailable=TRUE*)

END_VAR

VAR_OUTPUT
ReadData:ARRAY[0..255] OF BYTE;
ReadDataAvailable:BOOL;
Error:DWORD;
END_VAR

VAR
state:BYTE := 0;
ADSREAD:ADSREAD;
END_VAR
```

```
CASE
state OF
0:
        IF StartReading THEN
           ReadDataAvailable := FALSE;
           Error := 0;
           ADSRead(
               NETID:= ADSNetID,
               PORT:= PortNr,
               IDXGRP:= CO_Index,
               IDXOFFS:= CO_SubIndex,
               LEN:= DataLength,
               DESTADDR:= ADR(ReadData),
               READ:= TRUE,
               TMOUT := T#1s
               );
           IF ADSRead.err THEN
               state := 2;
               ReadDataAvailable := TRUE;
               Error := ADSRead.ErrId;
           ELSE
               state := 1;
           END_IF
        ELSE
           ADSRead(
               NETID:= ADSNetID,
               PORT:= PortNr,
               IDXGRP:= CO_Index,
               IDXOFFS:= CO_SubIndex,
```

```
                LEN:= DataLength,
                DESTADDR:= ADR(ReadData),
                READ:= FALSE,
                TMOUT := T#1s
            );
        END_IF
    1:
        ADSRead(READ:= FALSE);
        IF ADSRead.err THEN
            state := 2;
            ReadDataAvailable := TRUE;
            Error := ADSRead.ErrId;
        ELSE
            IF NOT ADSRead.busy THEN
                state := 2;
                ReadDataAvailable := TRUE;
            END_IF
        END_IF
    2:
        ADSRead(READ:= FALSE);
        state := 0;
END_CASE
```

Sample: SDO Write via ADS

In the following sample program (structured text) for the use of ADS services for SDO communication, object 0x6200, sub-index 3, from the node with port number 0x1001 is written. It concerns digital outputs to an I/O node.

```
(* Data to be written *)
WriteData[0] := 16#55;

(* write Object *)
SDO_WRITE(
StartWriting := WriteStart,
CO_Index := 16#6200,
CO_SubIndex := 3,
DataLength := 1,
PortNr := 16#1001,
WriteData := WriteData,
ADSNetID:='192.168.10.11.2.1'
);
IF SDO_WRITE.WriteDataFinished THEN
WriteStart := FALSE;
WriteError := SDO_WRITE.Error;
END_IF
```

The SDO_WRITE function block that has been called in turn calls the ADSWRITE function a number of times. It looks like this (starting with the variable declaration):

```
FUNCTION_BLOCK SDO_WRITE
VAR_INPUT

ADSNetID:STRING(23);    (* The AMSNetID addresses the EL6751. Can be empty if only one local single c
hannel card is present*)
PortNr:WORD;            (* The Port No. addresses the CANopen Node (see System Manager) *)
CO_Index:DWORD;         (* This is the Index of the CANopen Object Dictionary Entry*)
CO_SubIndex:DWORD;      (*This is the Sub-Index of the CANopen Object Dictionary Entry*)
DataLength:DWORD;       (* This is the Length of the CANopen Object Dictionary Entry*)
StartWriting:BOOL;      (*only reset to FALSE after WriteDataFinished=TRUE*)

WriteData:ARRAY[0..255] OF BYTE; (*This array contains the data to be written to the CANopen Object
Dictionary*)
END_VAR
VAR_OUTPUT
WriteDataFinished:BOOL;
Error:DWORD;
END_VAR
VAR
state:BYTE := 0;
ADSWRITE:ADSWRITE;
END_VAR
```

```
CASE
state OF
0:
        IF StartWriting THEN
           WriteDataFinished := FALSE;
           Error := 0;
           ADSWrite(
               NETID:= ADSNetID,
               PORT:= PortNr,
               IDXGRP:= CO_Index,
               IDXOFFS:= CO_SubIndex,
               LEN:= DataLength,
               SRCADDR:= ADR(WriteData),
               WRITE:= TRUE,
               TMOUT := T#1s
               );
           IF ADSWrite.err THEN
               state := 2;
               WriteDataFinished := TRUE;
               Error := ADSWrite.ErrId;
           ELSE
               state := 1;
           END_IF
        ELSE
           ADSWrite(
               NETID:= '',
               PORT:= PortNr,
               IDXGRP:= CO_Index,
               IDXOFFS:= CO_SubIndex,
               LEN:= DataLength,
               SRCADDR:= ADR(WriteData),
               WRITE:= FALSE,
               TMOUT := T#1s
               );
        END_IF
1:
        ADSWrite(WRITE:= FALSE);
        IF ADSWrite.err THEN
           state := 2;
           WriteDataFinished := TRUE;
           Error := ADSWrite.ErrId;
        ELSE
           IF NOT ADSWrite.busy THEN
               state := 2;
               WriteDataFinished := TRUE;
           END_IF
        END_IF
2:
        ADSWrite(WRITE:= FALSE);
        state := 0;
END_CASE
```

5.4.7 CANopen baud rate and bit timing

Bit Timing

The following baud rates and entries in the bit-timing register are supported by the CANopen devices:

Baud rate [kbaud]	BTR0	BTR1	Sampling Point
1000	0x00	0x14	75%
800	0x00	0x16	80%
500	0x00	0x1C	87%
250	0x01	0x1C	87%
125	0x03	0x1C	87%
100	0x04	0x1C	87%
50	0x09	0x1C	87%
20	0x18	0x1C	87%
10	0x31	0x1C	87%

The bit-timing register settings given (BTR0, BTR1) apply, for example, for the Philips 82C200, SJA1000, Intel 80C527, Siemens 80C167 and other CAN controllers. They are optimized for the maximum bus length.

5.4.8 Identifier Allocation

Default identifier

CANopen provides default identifiers for the most important communication objects, and these are derived from the 7-bit node address (the node ID) and a 4-bit function code in accordance with the following scheme:

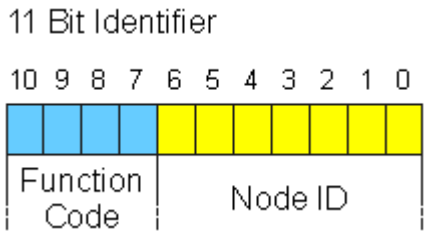


Fig. 124: Schematic: CANopen default identifier

For broadcast objects the node ID is set to *0*. This gives rise to the following default identifiers:

Broadcast objects

Object	Function	Function code	Resulting COB ID hex / dec	Object for communication Parameter / mapping
NMT	Boot-Up	0	0x00 / 0	- / -
SYNC	Synchronization	1	0x80 / 128	0x1005 + 0x1006 / -

Peer-to-peer objects

Object	Function with I/O devices	Function code	Resulting COB ID hex / dec	Object for communication Parameter / mapping
Emergency	Status / error	1	0x81 - 0xFF/ 129 - 255	- / -
PDO1 (tx)	dig. inputs	11	0x181 - 0x1FF / 385 - 511	0x1800 / 0x1A00
PDO1 (rx)	digital outputs	100	0x201 - 0x27F/ 513-639	0x1400 / 0x1600
PDO2 (tx)	analog inputs	101	0x281 - 0x2FF/ 641-767	0x1801 / 0x1A01
PDO2 (rx)	analog outputs	110	0x301 - 0x37F/ 769-895	0x1401 / 0x1601
PDO3 (tx)	analog inputs*	111	0x381 - 0x3FF / 897 - 1023	0x1802 / 0x1A02
PDO3 (rx)	analog outputs*	1000	0x401 - 0x47F/ 1025 - 1151	0x1402 / 0x1602
PDO4 (tx)	analog inputs*	1001	0x481 - 0x4FF/ 1153 - 1279	0x1803 / 0x1A03
PDO4 (rx)	analog outputs*	1010	0x501 - 0x57F/ 1281 - 1407	0x1403 / 0x1603
SDO (tx)	Parameter	1011	0x581 - 0x5FF/ 1409-1535	- / -
SDO (rx)	Parameter	1100	0x601 - 0x67F/ 1537-1663	- / -
Guarding	Life and node guarding, heartbeat, boot-up message	1110	0x701 - 0x77F/ 1793-1919	(0x100C, 0x100D, 0x100E, 0x1016, 0x1017)

* For historical reasons, the Beckhoff default mapping applies to PDOs 3 and 4 in Beckhoff I/O devices. In most configurations, PDOs 3 and 4 contain data related to analog inputs and outputs, but there can also be "excess" data from digital I/Os, or data from special terminals. Details may be found in the Bus Coupler documentation.

Up until version 3 of the CANopen specification, default identifiers were assigned to 2 PDOs at a time. The Beckhoff Bus Couplers up to firmware status BA correspond to this issue of the specification. After firmware status C0 (CANopen version 4), default identifiers are provided for up to 4 PDOs.

5.4.9 Firmware versions

Notes on Firmware 18 (FW 18 [▶ 176])

- By default, the EL6751 does not start transmitting the RxPDOs until one second after sending the Startup NMT message to a CANopen node. This function can be deactivated in the "Advanced Settings" (see fig.). In this case, immediately after the startup NMT message, the EL6751 starts sending the RxPDo for that node.

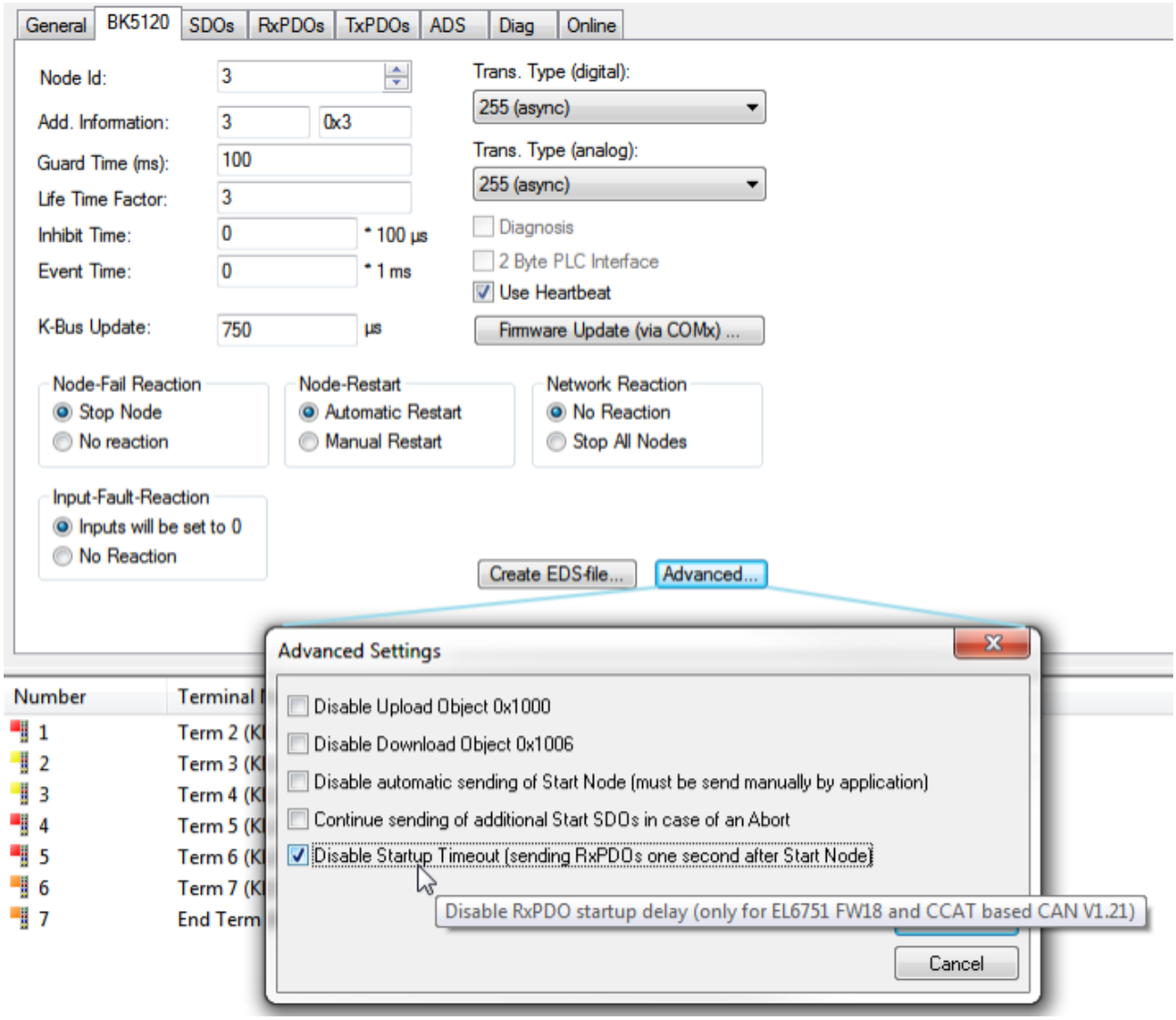


Fig. 125: Advanced Settings tab

- The terminal now also allows configuration of CANopen nodes without PDOs. These can still be used with asynchronous services.
- The option of sending any CAN messages via the IndexGroup 0xF923 has been extended by a function of receiving CAN messages.

5.4.10 Sending and receiving of CAN Messages (STD Frame Format) via ADS

Sending CAN messages via ADS

```
AdsWrite:

NETID = AoeNetId der EL6751
PORT = 200
IDXGRP = 16#F921
IDXOFFS = 0
LEN = Length of the following DATA,

DATA[0]: 1st CAN-Message, CAN-Id Bit 0-7
DATA[1], Bit 0-2: 1st CAN-Message, CAN-Id Bit 8-10
```

```
DATA[1], Bit 7: 1 = RTR (the length then indicates the number of data to be read, no data follows
the message, the next CAN message starts at DATA [3]
DATA[2]: Length of the 1st CAN-Message
DATA[3-n]: Data of the 1st CAN-Message

DATA[(n+1)]: 2nd CAN-Message, CAN-Id Bit 0-7
etc.
```

Enable / disable CAN messages for receiving

For receiving, the CAN IDs must first be activated.

```
AdsWrite:

NETID = AoeNetId der EL6751
PORT = 200
IDXGRP = 16#F923
IDXOFFS = 0
LEN = Number of CAN-IDs * 2

DATA[0]: 1st CAN-ID, Bit 0-7
DATA[1]: 1st CAN-ID, Bit 8-11,
Bit 15=0: Activate for receiving
Bit 15=1: Deactivate for receiving
DATA[2]: 2nd CAN-ID, Bit 0-7
etc.
```

Reading the received CAN messages

```
AdsRead:

NETID = AoeNetId der EL6751
PORT = 200
IDXGRP = 16#F921
IDXOFFS = 0
LEN = 640 (maximum buffer size)

The DATA has the same structure when sending the CAN messages.
```

The buffer in the EL6751 comprises approx. 50 CAN messages (with 8 bytes of data per frame).
This feature is available from FW18.

5.5 EtherCAT communication EL6751

5.5.1 CANopen master

5.5.1.1 EtherCAT State Machine

The EL6751 can be configured in several ways:

1. Configuration of the EL6751 with StartUp SDOs [▶ 123]: Here, the StartUp SDOs are calculated in the EtherCAT configurator and transferred to the EtherCAT master, in the same way as is carried out, for example, in the TwinCAT System Manager.

2. Configuration of the EL6751 by scanning the CAN bus [▶ 124]: Here, the EL6751 is ordered to scan the CAN bus and to save the CANopen configuration found there in the InfoData objects.

3. Configuration of the EL6751 with Backup Parameter Storage [▶ 126]: Here, the configuration of the CANopen slave is stored in the flash memory of the EL6751 and need only be transmitted once.

Configuration of the EL6751 with StartUp SDOs

The following flow chart shows the sequence of the configuration of the EL6751 with StartUp SDOs:

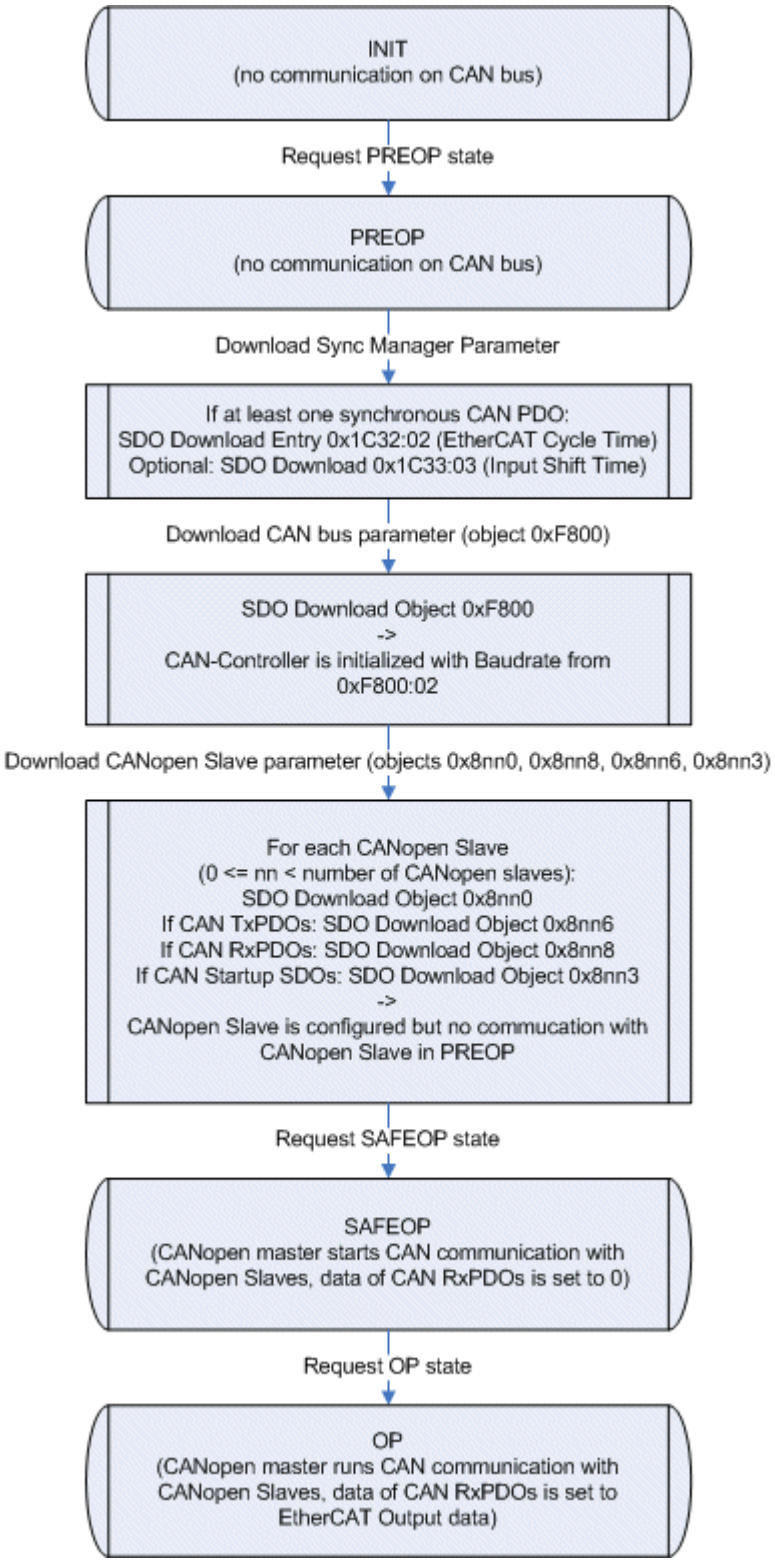


Fig. 126: Flow chart for the EL6751 with Start SDOs

After a power-on, the EL6751 is in the INIT state and has no CANopen configuration. The CAN controller is in the OFFLINE state.

CAN bus parameters

The CANopen configuration is carried out via SDO download in the PREOP state. The objects to be loaded must be transmitted either with Complete Access or with consistency nesting (first set SubIndex 0 to 0, then write SubIndex 1-n, then set SubIndex 0 to n). Care should thereby be taken to always start with object 0xF800. After receiving the object 0xF800, the EL6751 switches the CAN controller with the appropriate baud rate from 0xF800:02 to ONLINE.

CANopen slave configuration

After object 0xF800, the objects 0x8yy0, 0x8yy6 (if the CANopen slave has CAN TxPDOs), 0x8yy8 (if the CANopen slave has CAN RxPDOs) and 0x8yy3 (if application-specific StartUp-SDOs are to be sent to the CANopen slave before the sending of the CAN Start Node command) must be transmitted in this order for each CANopen slave to be configured. yy is to be incremented (starting from 0) for each CANopen slave to be configured.

PDO Mapping

For each configured CANopen slave, there is an EtherCAT RxPDO (if the CANopen slave has CAN RxPDOs) and an EtherCAT TxPDO (if the CANopen slave has CAN TxPDOs). The PDO mapping of the EtherCAT PDOs is automatically calculated by the EL6751 after the download of the respective 0x8yyz objects and can be read. The PDO mapping objects 0x16yy and 0x1Ayy thereby belong to the configuration objects 0x8yyz. The PDO mapping objects can only be written with the values that the EL6751 has calculated itself. The writing of the PDO mapping thus serves only to check the PDO mapping calculated by the EtherCAT configurator and can therefore be omitted.

PDO Assign

In addition, there are a few other EtherCAT PDOs that contain control, status and diagnostic information. These PDOs are selected via the PDO Assign. It should thereby be ensured that all EtherCAT PDOs that are assigned to the configured CANopen slaves (PDO number <= 128) always appear in the PDO Assign. With regard to the order of the PDOs in the PDO Assign, it is important to ensure that the index of the assigned EtherCAT PDO increases with each entry in the corresponding PDO Assign object. If the EtherCAT master does not transmit any PDO Assign in the StartUp SDOs, then PDOs 0x1A83 and 0x1A85 are assigned for status and diagnosis.

Cyclic CANopen communication

During the transition to SAFEOP, the EL6751 checks the length configured in the Sync Manager channels 2 and 3 against the length calculated from PDO Mapping and PDO Assign. The SAFEOP state is only adopted if these lengths match. In the SAFEOP state, the EL6751 starts the boot-up of the configured CANopen slaves. After the transmission of all CAN StartUp SDOs, the respective CANopen slave is started with the ‘Start Node’ message and the CAN PDO communication is active. All outputs in the CAN RxPDOs are thereby set to 0. As soon as the EL6751 has been switched to OP, the data from the EtherCAT outputs are also adopted into the CAN RxPDOs.

Configuration of the EL6751 by scanning the CAN bus

The following flow chart shows the sequence of the configuration of the EL6751 by scanning the CAN bus:

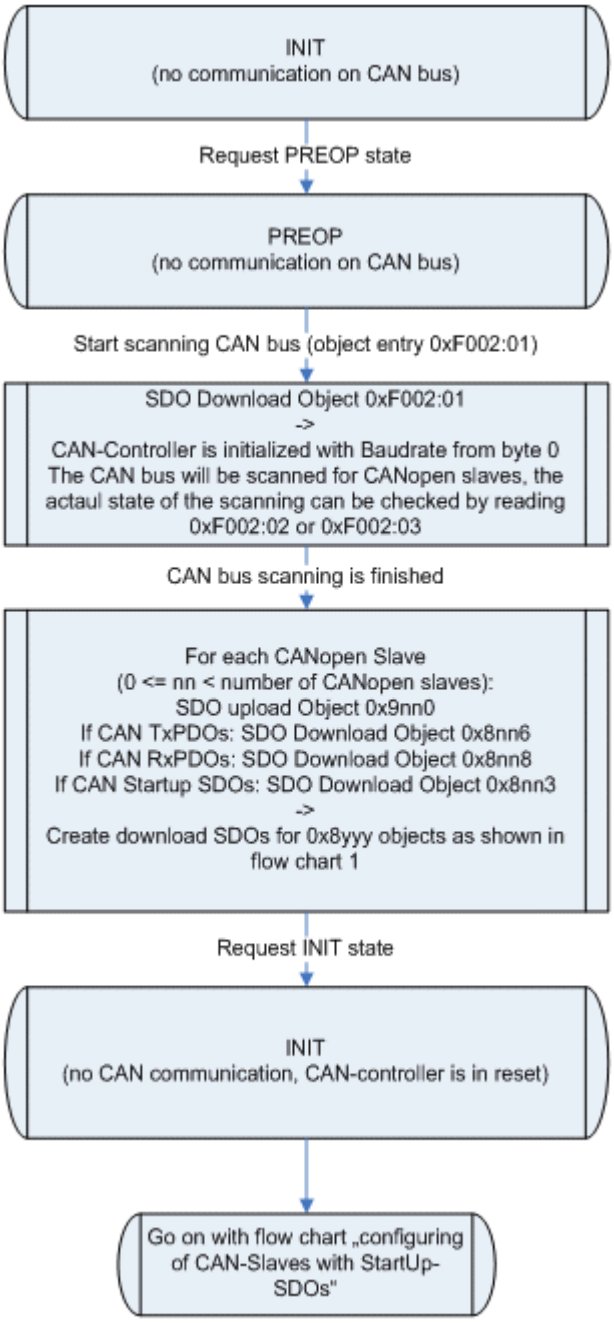


Fig. 127: Flow chart for EL6751 with scanning of the CAN bus

After a power-on, the EL6751 is in the INIT state and has no CANopen configuration. The CAN controller is in the OFFLINE state.

Scanning the CAN bus

The scanning of the CAN bus can be started in the PREOP state, provided that no CANopen configuration has been loaded yet. To this end, the desired baud rate must be written to entry 0xF002:01 (SDO download). The EL6751 switches the CAN controller with the appropriate baud rate to ONLINE and scans the CAN bus. By cyclically reading (SDO upload) the entries 0xF002:02 or 0xF002:03, the EtherCAT master can determine the progress of the scanning of the CAN bus.

Reading the CANopen slave configuration

After completion of the scan, the InfoData objects 0x9yy0, 0x9yy8 and 0x9yyA contain the CANopen slave configuration found. The number of CANopen slaves found can be read via entry 0xF002:03 or object 0xF040. The EtherCAT master can now read the InfoData objects 0x9yyz, generate the StartUp objects 0x8yyz from them and proceed according to the Configuration of the EL6751 with StartUp SDOs [▶ 123].

Creating the Backup Parameter Storage

As an alternative to reading the InfoData, the Backup Parameter Storage can also be created by writing the value 0x65766173 to entry 0x1010:01. Subsequently, the EL6751 must be switched to INIT and with configuration

Configuration of the EL6751 with Backup Parameter Storage

The following flow chart shows the sequence of the configuration of the EL6751 with Backup Parameter Storage:

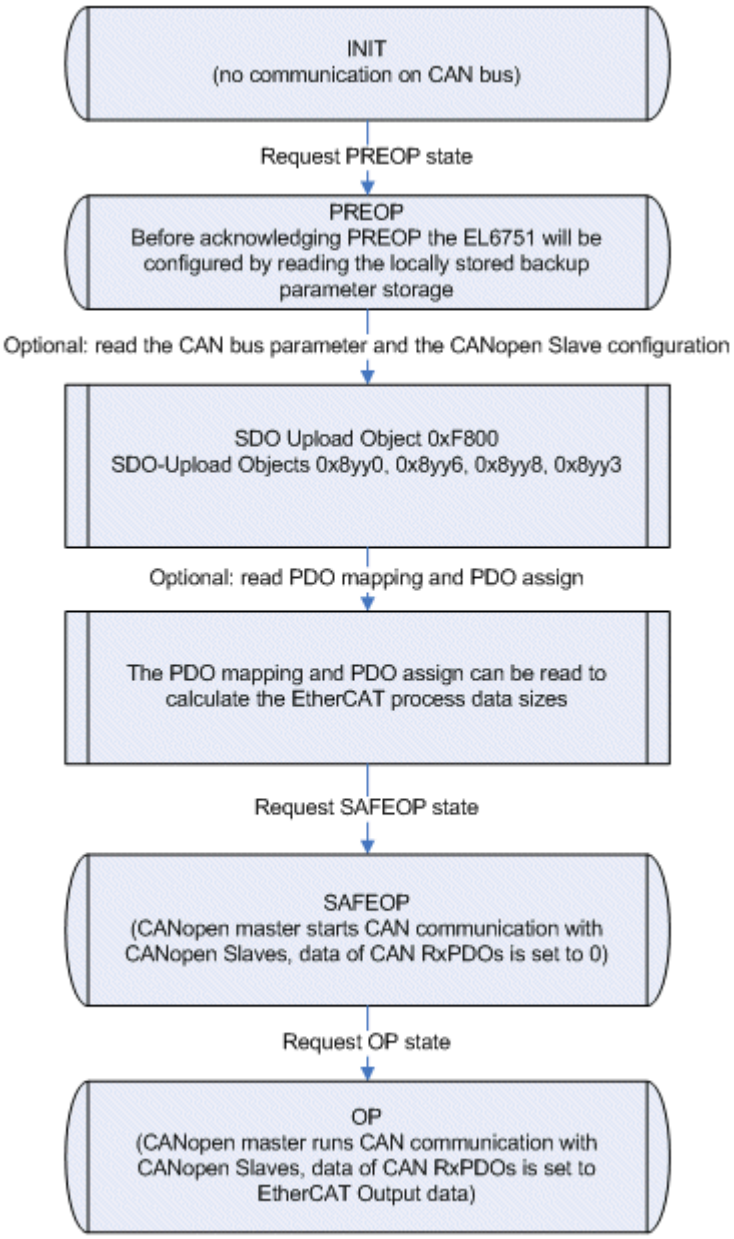


Fig. 128: Flow chart for EL6751 with Backup Parameter Storage

After a power-on, the EL6751 is in the INIT state and has no CANopen configuration. The CAN controller is in the OFFLINE state.

CAN Bus parameter / CANopen slave configuration

The configuration stored in the Backup Parameter Storage object 0x10F2 is loaded during the transition from INIT to PREOP. Since the StartUp SDOs from the Configuration of the EL6751 with StartUp SDOs [▶ 123] are stored in the Backup Parameter Storage object, the sequence is the same as the one described there. Hence, the stored data are first written to object 0xF800 and then the EL6751 switches the CAN controller with the appropriate baud rate from 0xF800:02 to ONLINE. Subsequently, the CANopen slaves are generated according to the stored CANopen slave configuration. If the PREOP state is acknowledged, the current CANopen configuration in objects 0xF800, 0x8yy0, 0x8yy6, 0x8yy8 and 0x8yy3 can be read.

PDO Mapping / PDO Assign

In addition, the EtherCAT master can also read the PDO Mapping and PDO Assign in the PREOP state in order to determine the lengths of the EtherCAT process data.

Creating the Backup Parameter Storage

The Backup Parameter Storage can be created as follows:

1. Download the object 0x10F2 (in PREOP, SAFEOP or OP): in this case, the data received are stored as Backup Parameter Storage in the flash memory

2. Scan the CAN bus and then write the value 0x65766173 to entry 0x1010:01: here, the EL6751 automatically generates the Backup Parameter Storage from the InfoData 0x9yy0, 0x9yy8 and 0x9yyA and stores it in the flash memory.

In both cases the EL6751 is automatically rebooted after 5 s (reverts to INIT with the AL status code 0x60). Furthermore, all SDO downloads of the objects 0xF800 or 0x8yyz are rejected.

Deleting the Backup Parameter Storage

In order to load a new Backup Parameter Storage or to simply delete the existing one, the value 0x64616F6C has to be written to entry 0x1011:01.

5.5.1.2 Synchronization

In the EL 6751, the CAN cycle is synchronized with the EtherCAT cycle. Synchronization takes place by default via the Sync Manager 2 event or, if there is no EtherCAT output process data, via the Sync Manager 3 event. Alternatively, the EL6751 can also be operated in the Distributed Clocks mode, in which case synchronization takes place via the SNYC0 and SYNC1 events.

CAN cycle (Sync Multiplier = 1)

The following flow chart shows the sequence of the CAN cycle if a CAN cycle is also executed with each EtherCAT cycle (Sync Multiplier = 1 (0x1C32:02 == 0xF800:04 or 0x1C32:02 == 0 (default) or 0xF800:04 == 0)). If the cycle time of the EtherCAT master (0x1C32:05) is not transmitted in the StartUp SDOs (or the Backup Parameter Storage object) and no Distributed Clocks mode is set, then 0x1C32:05 = 0 and, hence, the Sync Multiplier = 1.

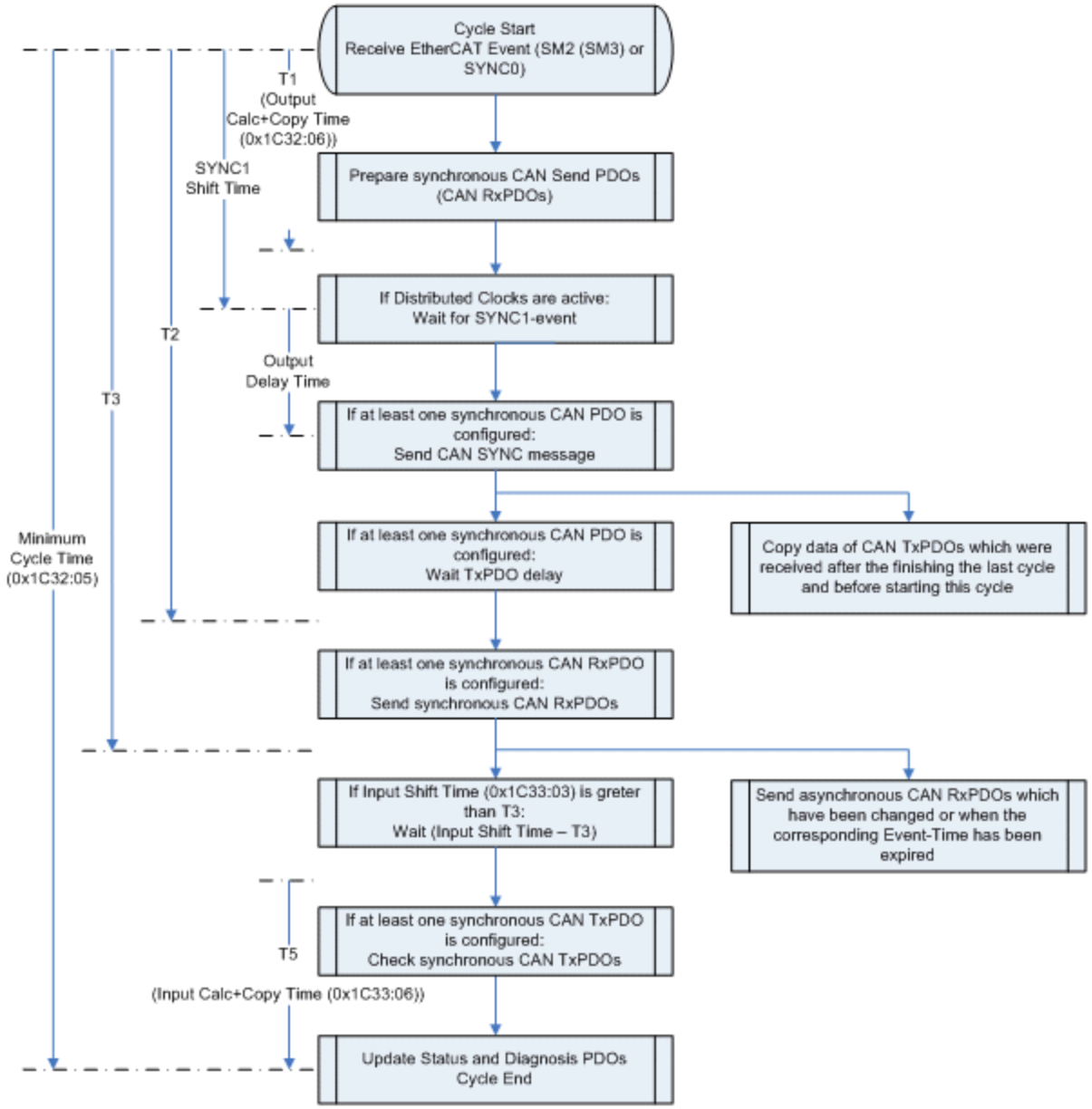


Fig. 129: Flow chart for CAN cycle (Sync Multiplier = 1)

Synchronization with SM2 (SM3) event

When receiving the EtherCAT process data telegram, the SM2 event (SM3 if no EtherCAT output data are configured, i.e. only CANopen slaves without CAN RxPDOs) is generated by the EtherCAT slave controller, thus starting the CAN cycle. If synchronous CAN PDOs are configured, they will be dealt with at the beginning. Following the preparation of CAN transmit PDOs, the SYNC message will initially be sent. Because some CANopen slaves react strangely if their RxPDOs are received before they have sent their TxPDOs, a delay can be set for the TxPDOs in 0xF800:0E. After sending the SYNC message, the EL6751 waits until this delay has expired before sending any further CAN messages. After that, the synchronous RxPDOs are transmitted first, followed by the asynchronous RxPDOs (if they have changed or if the event time has expired). If the synchronous RxPDOs have been sent, the expiry of the input shift time is waited for. Subsequently, the receipt of the synchronous TxPDOs is checked. If the transmission type of a TxPDO is set to 1, the EL6751 expects a RxPDO in each cycle until the time T4; if this has not been received, the Node State of the CANopen slave (0xF102:yy) is set to 0x28 for one cycle. If the next SM2 (SM3) event is received before the CAN cycle is completed, the Cycle Exceed counter (0x1C32:0B or 0x1C33:0B) is incremented and a CAN cycle is omitted.

If only asynchronous PDOs are configured, then the process is much simpler. After receiving the SM2 event, the asynchronous CAN RxPDOs to be sent are determined (data has changed or event time has expired) and sent. Then the CAN TxPDOs received since the end of the last cycle are copied to the EtherCAT input data. If the Sync Managers 2 and 3 are set to 1-buffer mode and the input shift time is not equal to 0, the copying of the status and diagnosis to the EtherCAT input data is delayed until this time has expired. The advantage of this is that CAN TxPDOs received within this time window can be copied directly into the EtherCAT input data. However, care must be taken when setting the input shift time to ensure that the EtherCAT input data can be copied completely before the next EtherCAT cycle begins, otherwise the working the counter would not be okay.

Synchronization with SYNC0/SYNC1 event

The Distributed Clocks mode only makes sense if synchronous CAN PDOs are present. In this case, the CAN cycle is started by the SYNC0 event. The sending of the SYNC message is delayed until the SYNC1 event occurs, so that the SYNC message is sent with a jitter of maximum 500 ns. The remaining sequence of the CAN cycle corresponds to that in the case of synchronization with SM2 event.

CAN cycle (Sync Multiplier > 1)

The following flow chart shows the sequence of the CAN cycle if a CAN cycle is also executed with every n^{th} EtherCAT cycle (n > 1) (Sync Multiplier > 1 (n*0x1C32:02 == 0xF800:04 and 0x1C32:02 != 0 and 0xF800:04 != 0)). If no Distributed Clocks mode is set, then the cycle time of the EtherCAT master (0x1C32:02) must be transmitted in the StartUp SDOs (or the Backup Parameter Storage object).

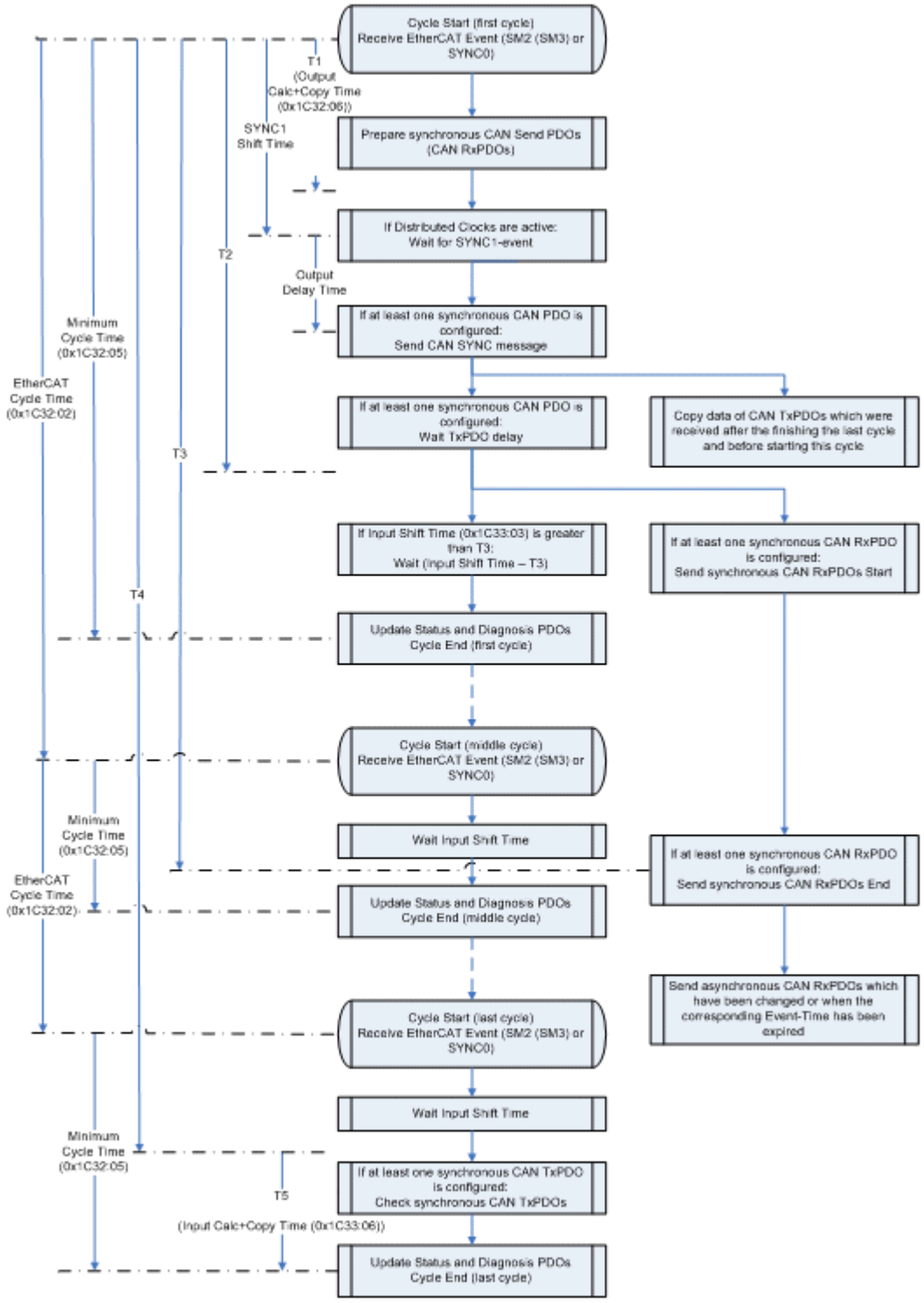


Fig. 130: Flow chart for CAN cycle (Sync Multiplier > 1)

Synchronization with SM2 (SM3) event

Whereas in operating mode Sync Multiplier = 1 the synchronous RxPDOs must be sent completely before the EtherCAT input data is updated, this can extend over n EtherCAT cycles in operation with Sync Multiplier = n. The beginning of the CAN cycle is still the same as in operation with Sync Multiplier = 1. After the TxPDO delay has expired (on the flow chart it is assumed that the TxPDO delay is smaller than the input shift time), transmission of the synchronous RxPDOs begins. If the input shift time expires during that time, the EtherCAT input data containing the most recent data from all TxPDOs received up to this time is updated. After that, the next event is waited for each time until the last EtherCAT cycle within the CAN cycle is reached. In the middle EtherCAT cycles, the data of the asynchronous RxPDOs to be sent always receive the latest value, whereas the data of the synchronous RxPDOs are only updated in the first EtherCAT cycle.

In the last EtherCAT cycle, the synchronous RxPDOs are checked following the expiry of the input shift time (as in operation with Sync Multiplier = 1). A Sync Multiplier > 1 is ineffective if only asynchronous PDOs are configured.

Synchronization with SYNC0/SYNC1 event

If distributed clocks are switched on, the CAN cycle would be started by the SYNC0 event. The sending of the SYNC message is delayed until the SYNC1 event occurs, so that the SYNC message is sent with a jitter of maximum 500 ns. The remaining sequence of the CAN cycle corresponds to that in the case of synchronization with SM2 event.

5.5.1.3 Object description and parameterization

EtherCAT XML Device Description

The display matches that of the CoE objects from the EtherCAT XML Device Description. We recommend downloading the latest XML file from the download area of the Beckhoff website and installing it according to installation instructions.

Parameterization via the CoE list (CAN over EtherCAT)

The EtherCAT device is parameterized via the CoE-Online tab (double-click on the respective object) or via the Process Data tab (allocation of PDOs). Please note the following general CoE notes [▶ 37] when using/manipulating the CoE parameters:

- Keep a startup list if components have to be replaced
- Differentiation between online/offline dictionary, existence of current XML description
- use “CoE reload” for resetting changes

Presentation of the object description with "MDP Mapping“

The following object description uses the "Modular Device Profile Mapping" (MDP) for the mapping of the terminal information to the process image.

5.5.1.3.1 Standard objects (0x1000-0x1FFF)

The standard objects have the same meaning for all EtherCAT slaves.

Index 1000 Device type

Index (hex)	Name	Meaning	Data type	Flags	Default
1000:0	Device type	Device type of the EtherCAT slave: The Lo-Word contains the CoE profile used (5001). The Hi-Word contains the module profile according to the modular device profile.	UINT32	RO	0x00001389 (5001_{dec})

Index 1008 Device name

Index (hex)	Name	Meaning	Data type	Flags	Default
1008:0	Device name	Device name of the EtherCAT slave	STRING	RO	EL6751

Index 1009 Hardware version

Index (hex)	Name	Meaning	Data type	Flags	Default
1009:0	Hardware version	Hardware version of the EtherCAT slave	STRING	RO	

Index 100A Software version

Index (hex)	Name	Meaning	Data type	Flags	Default
100A:0	Software version	Firmware version of the EtherCAT slave	STRING	RO	

Index 1010 Store parameters

Index (hex)	Name	Meaning	Data type	Flags	Default
1010:0	Store parameters	Save the CANopen configuration after scanning the CAN bus with entry 0xF002:01	UINT8	RO	
1010:01	SubIndex 001	If you set this entry to **'0x65766173'**, the Backup Parameter Storage (object 0x10F2 [▶ 133]) is generated from the InfoData 0x9yyz.	UINT32	RW	

Index 1011 Restore default parameters

Index (hex)	Name	Meaning	Data type	Flags	Default
1011:0	Restore default parameters	Restore default parameters	UINT8	RO	
1011:01	SubIndex 001	If you set this entry to **'0x64616F6C'**, all backup objects are reset to the delivery state.	UINT32	RW	

Index 1018 Identity

Index (hex)	Name	Meaning	Data type	Flags	Default
1018:0	Identity	Information for identifying the slave	UINT8	RO	0x04 (4_{dec})
1018:01	Vendor ID	Vendor ID of the EtherCAT slave	UINT32	RO	0x00000002 (2_{dec})
1018:02	Product code	Product code of the EtherCAT slave	UINT32	RO	0x1A5F3052 (442445906_{de}C)
1018:03	Revision	Revision number of the EtherCAT slave; the low word (bit 0-15) indicates the special terminal number, the high word (bit 16-31) refers to the device description	UINT32	RO	0x00100000 (1048576_{dec})
1018:04	Serial number	Serial number of the EtherCAT slave; the low byte (bit 0-7) of the low word contains the year of production, the high byte (bit 8-15) of the low word contains the week of production, the high word (bit 16-31) is 0	UINT32	RO	0x00000000 (0_{dec})

Index 10F0 Backup parameter handling

Index (hex)	Name	Meaning	Data type	Flags	Default
10F0:0	Backup parameter handling	Information for standardized loading and saving of backup entries	UINT8	RO	
10F0:01	Checksum	Checksum of the Backup Parameter Storage (object 0x10F2 [▶ 133], word 2-3)	UINT32	RO	

Index 10F2 Backup parameter storage

Index	Name	Meaning		Data type	Flags	Default
10F2:0	Backup parameter storage	If this object is used, no StartUp SDOs may be transmitted in the PREOP state, since the Backup Parameter Storage contains the complete StartUp SDOs (see Configuration of the EL6751 with Backup Parameter Storage [▶ 126]). The EL6751 is rebooted 5 s after the flashing of the Backup Parameter Storage (switches to INIT with AL status code = 0x60). The data have the following meaning:		OCTET-STRING[n]	RW	
		Word-Offset	Description			
		0	Command: with 0xC0DE, the received data are stored in the flash memory			
		1	Length of the data from word offset 4 in bytes			
		2-3	Checksum, which is calculated locally			
		4	Index of the object of the 1st StartUp SDO			
		5	len1: Length of the object of the 1st StartUp SDO as CompleteAccess (from SubIndex 0) in bytes			
		6-n1	Data of the object of the 1st StartUp SDO as CompleteAccess (n1 = 2*((len1+1)/2)+5)			
		n1+1	Index of the object of the 2nd StartUp SDO			
		n1+2	len2: Length of the object of the 2nd StartUp SDO as CompleteAccess (from SubIndex 0) in bytes			
		(n1+3)-n2	Data of the object of the 2nd StartUp SDO as CompleteAccess (n2 = 2*((len2+1)/2)+n1+2)			
		...				
		m	Index of the object of the 3rd StartUp SDO			
		m+1	len3: Length of the object of the 3rd StartUp SDO as CompleteAccess (from SubIndex 0) in bytes			
		(m+2)-n3	Data of the object of the 3rd StartUp SDO as CompleteAccess (n3 = 2*((len3+1)/2)+m+1)			

Index 1600-167E RxPDO-Map Node yyy

Index (hex)	Name	Meaning	Data type	Flags	Default
1600+n:0	RxPDO-Map Node yyy	For each configured CANopen slave there is one RxPDO, which contains all CAN RxPDOs of the CANopen slave. The CAN RxPDOs written in object 0x8008 [▶ 146]+ (n*16) are located in the RxPDO mapping object 0x1600+n. If a CANopen slave contains no CAN RxPDOs, then neither object 0x8008 [▶ 146]+16*n nor the PDO mapping object 0x1600+n exist. These PDOs are mandatory and must always be contained in the PDO Assign object 0x1C12 [▶ 139], depending on the configured CANopen slaves. SubIndex 0 contains the number of CAN RxPDOs of the (n+1)th configured CANopen slave. The RxPDO mapping objects 0x1600-0x167E can be written in order to change the order of the CAN RxPDOs of a configured CANopen slave within its EtherCAT RxPDO. If a RxPDO mapping object of the EtherCAT RxPDOs 1-127 is written, then all PDO mapping objects of the EtherCAT RxPDOs 1-127 and the EtherCAT TxPDOs 1-127 must always be written.	UINT8	RW	
(1600+n):01		first mapped CAN RxPDO of the (n+1)th configured CANopen slave	UINT32	RW	
...		..			
(1600+n):m		last mapped CAN RxPDO of the (n+1)th configured CANopen slave	UINT32	RW	

Index 1685 RxPDO-Map Control

Index (hex)	Name	Meaning	Data type	Flags	Default
1685:0	RxPDO-Map Control	The control word (index 0xF200 [▶ 153]) can be mapped into the EtherCAT output data with this PDO. This PDO is optional.	UINT8	RO	0x02 (2_{dec})
1685:01	SubIndex 001	1. PDO Mapping entry (object 0xF200 (Control), entry 0x01 (CAN Controller Auto Reset when BUS-OFF))	UINT32	RO	0xF200:01, 1
1685:02	SubIndex 002	2. PDO Mapping entry (15 bits align)	UINT32	RO	0x0000:00, 15

Index 1881 TxPDO-Par PDO State

Index (hex)	Name	Meaning	Data type	Flags	Default
1881:0	TxPDO-Par PDO State	PDO Parameter TxPDO 130	UINT8	RO	0x06 (6_{dec})
1881:06	Exclude TxPDOs	Specifies the TxPDOs (index of TxPDO mapping objects) that must not be transferred together with TxPDO 130	OCTET-STRING[14]	RO	80 1A 00 00 00 00 00 00 00 00 00 00 00 00

Index 1882 TxPDO-Par CANopen Diag Flag

Index (hex)	Name	Meaning	Data type	Flags	Default
1882:0	TxPDO-Par CANopen Diag Flag	PDO parameter TxPDO 131	UINT8	RO	0x06 (6_{dec})
1882:06	Exclude TxPDOs	Specifies the TxPDOs (index of TxPDO mapping objects) that must not be transferred together with TxPDO 131	OCTET-STRING[14]	RO	80 1A 00 00 00 00 00 00 00 00 00 00 00 00

Index 1883 TxPDO-Par Node State

Index (hex)	Name	Meaning	Data type	Flags	Default
1883:0	TxPDO-Par Node State	PDO parameter TxPDO 132	UINT8	RO	0x06 (6_{dec})
1883:06	Exclude TxPDOs	Specifies the TxPDOs (index of TxPDO mapping objects) that must not be transferred together with TxPDO 132	OCTET-STRING[14]	RO	80 1A 00 00 00 00 00 00 00 00 00 00 00 00

Index 1884 TxPDO-Par Extended Diag

Index (hex)	Name	Meaning	Data type	Flags	Default
1884:0	TxPDO-Par Extended Diag	PDO parameter TxPDO 133	UINT8	RO	0x06 (6_{dec})
1884:06	Exclude TxPDOs	Specifies the TxPDOs (index of TxPDO mapping objects) that must not be transferred together with TxPDO 133	OCTET-STRING[14]	RO	80 1A 85 1A 00 00 00 00 00 00 00 00 00 00

Index 1885 TxPDO-Par CAN Status

Index (hex)	Name	Meaning	Data type	Flags	Default
1885:0	TxPDO-Par CAN status	PDO parameter TxPDO 134	UINT8	RO	0x06 (6_{dec})
1885:06	Exclude TxPDOs	Specifies the TxPDOs (index of TxPDO mapping objects) that must not be transferred together with TxPDO 134	OCTET-STRING[14]	RO	80 1A 84 1A 00 00 00 00 00 00 00 00 00 00

Index 1888 TxPDO-Par CAN TxPDO Toggle 1

Index (hex)	Name	Meaning	Data type	Flags	Default
1888:0	TxPDO-Par CAN TxPDO Toggle 1	PDO parameter TxPDO 137	UINT8	RO	0x06 (6_{dec})
1888:06	Exclude TxPDOs	Specifies the TxPDOs (index of TxPDO mapping objects) that must not be transferred together with TxPDO 137	OCTET-STRING[14]	RO	80 1A 00 00 00 00 00 00 00 00 00 00 00 00

Index 1889 TxPDO-Par CAN TxPDO Toggle 2

Index (hex)	Name	Meaning	Data type	Flags	Default
1889:0	TxPDO-Par CAN TxPDO Toggle 2	PDO parameter TxPDO 138	UINT8	RO	0x06 (6_{dec})
1889:06	Exclude TxPDOs	Specifies the TxPDOs (index of TxPDO mapping objects) that must not be transferred together with TxPDO 138	OCTET-STRING[14]	RO	80 1A 00 00 00 00 00 00 00 00 00 00 00 00

Index 1A00-1A7E TxPDO-Map Node yyy

Index (hex)	Name	Meaning	Data type	Flags	Default
1A00+n:0	TxPDO-Map Node yyy	For each configured CANopen slave there is one TxPDO, which contains all CAN TxPDOs of the CANopen slave. The CAN TxPDOs written in object 0x8006 [▶ 145]+ (n*16) are located in the TxPDO mapping object 0x1A00+n. If a CANopen slave contains no CAN TxPDOs, then neither object 0x8006 [▶ 145]+16*n nor the PDO mapping object 0x1A00+n exist. These PDOs are mandatory and must always be contained in the PDO Assign object 0x1C13 [▶ 139], depending on the configured CANopen slaves. SubIndex 0 contains the number of CAN RxPDOs of the (n+1)th configured CANopen slave. The TxPDO mapping objects 0x1A00-0x1A7E can be written in order to change the order of the CAN TxPDOs of a configured CANopen slave within its EtherCAT TxPDO. If a TxPDO mapping object of the EtherCAT TxPDOs 1-127 is written, then all PDO mapping objects of the EtherCAT TxPDOs 1-127 and the EtherCAT RxPDOs 1-127 must always be written.	UINT8	RW	
(1A00+n):01		first mapped CAN TxPDO of the (n+1)th configured CANopen slave	UINT32	RW	
...					
(1A00+n):m		last mapped CAN TxPDO of the (n+1)th configured CANopen slave	UINT32	RW	

Index 1A81 TxPDO-Map PDO State

Index (hex)	Name	Meaning	Data type	Flags	Default
1A81:0	TxPDO-Map PDO State	In this PDO there is a bit for each configured CANopen slave that is set if the CAN communication to the CANopen slave is not OK (a more detailed error cause can be found in 0xF102 [▶ 151]:m for the mth configured CANopen slave). If the bit is set, the data of the associated TxPDO m is to be ignored. This PDO is optional.	UINT8	RO	Number of configured CANopen slaves
1A81:01		PDO state of the first configured CANopen slave (configured via the objects 0x800z)	UINT32	RO	0x1800:07, 1
...					
1A81:m		PDO state of the last (mth) configured CANopen slave (configured via the objects 0x800z+(m-1)*16 (1 <= m <= 127))	UINT32	RO	0x1800+(m-1):07, 1

Index 1A82 TxPDO-Map CANopen Diag Flag

Index (hex)	Name	Meaning	Data type	Flags	Default
1A82:0	TxPDO-Map CANopen Diag Flag	In this PDO there is a bit for each configured CANopen slave that is set if the diagnostic information (object 0xF103 [▶ 152]) has changed. This PDO is optional.	UINT8	RO	Number of configured CANopen slaves
1A82:01		Diag Flag of the first configured CANopen slave (configured via the objects 0x800z)	UINT32	RO	0xF103:01,1
...					
1A82:m		Diag Flag of the last (m^{th}) configured CANopen slave (configured via the objects 0x800z+(m-1)*16 (1 <= m <= 127))	UINT32	RO	0xF103:m,1

Index 1A83 TxPDO-Map Node State

Index (hex)	Name	Meaning	Data type	Flags	Default
1A83:0	TxPDO-Map Node State	In this PDO there is a byte for each configured CANopen slave that contains the communication state (object 0xF102 [▶ 151]) to the CANopen slave. This PDO is optional.	UINT8	RO	0x00 (0_{dec})
1A83:01		Node state of the first configured CANopen slave (configured via the objects 0x800z)	UINT32	RO	0xF102:01,8
...					
1A83:m		Node state of the last (m^{th}) configured CANopen slave (configured via the objects 0x800z+(m-1)*16 (1 <= m <= 127))	UINT32	RO	0xF102:m,8

Index 1A84 TxPDO-Map Extended Diag

Index (hex)	Name	Meaning	Data type	Flags	Default
1A84:0	TxPDO-Map Extended Diag	This PDO contains the CAN status (object 0xF108 [▶ 152]) and the CANopen master diagnostics (object 0xF101 [▶ 151]) and is optional	UINT8	RO	0x16 (22_{dec})
1A84:01	SubIndex 001	1. PDO Mapping entry (11 bits align)	UINT32	RO	0x0000:00, B
1A84:02	SubIndex 002	2. PDO Mapping entry (object 0xF101 (Extended Diag), entry 0x0C (SYNC Toggle))	UINT32	RO	0xF101:0C, 1
1A84:03	SubIndex 003	3. PDO Mapping entry (object 0xF101 (Extended Diag), entry 0x0D (Device Diag))	UINT32	RO	0xF101:0D, 1
1A84:04	SubIndex 004	4. PDO Mapping entry (1 bits align)	UINT32	RO	0x0000:00, 1
1A84:05	SubIndex 005	5. PDO Mapping entry (object 0xF101 (Extended Diag), entry 0x0F (PDO Toggle))	UINT32	RO	0xF101:0F, 1
1A84:06	SubIndex 006	6. PDO Mapping entry (object 0xF101 (Extended Diag), entry 0x10 (PDO State))	UINT32	RO	0xF101:10, 1
1A84:07	SubIndex 007	7. PDO Mapping entry (object 0xF101 (Extended Diag), entry 0x11 (Cycle Counter))	UINT32	RO	0xF101:11, 16
1A84:08	SubIndex 008	8. PDO Mapping entry (object 0xF101 (Extended Diag), entry 0x12 (Slave Status Counter))	UINT32	RO	0xF101:12, 8
1A84:09	SubIndex 009	9. PDO Mapping entry (8 bits align)	UINT32	RO	0x0000:00, 8
1A84:0A	SubIndex 010	10. PDO Mapping entry (object 0xF101 (Extended Diag), entry 0x14 (Cycle Time))	UINT32	RO	0xF101:14, 16
1A84:0B	SubIndex 011	11. PDO Mapping entry (16 bits align)	UINT32	RO	0x0000:00, 16
1A84:0C	SubIndex 012	12. PDO Mapping entry (object 0xF108 (CAN Status), entry 0x21 (RX error counter))	UINT32	RO	0xF108:21, 8
1A84:0D	SubIndex 013	13. PDO Mapping entry (object 0xF108 (CAN Status), entry 0x22 (TX error counter))	UINT32	RO	0xF108:22, 8
1A84:0E	SubIndex 014	14. PDO Mapping entry (object 0xF108 (CAN Status), entry 0x01 (Bus-Off))	UINT32	RO	0xF108:01, 1
1A84:0F	SubIndex 015	15. PDO Mapping entry (object 0xF108 (CAN Status), entry 0x02 (warning limit reached))	UINT32	RO	0xF108:02, 1
1A84:10	SubIndex 016	16. PDO Mapping entry (object 0xF108 (CAN Status), entry 0x03 (RX overflow))	UINT32	RO	0xF108:03, 1
1A84:11	SubIndex 017	17. PDO Mapping entry (1 bits align)	UINT32	RO	0x0000:00, 1
1A84:12	SubIndex 018	18. PDO Mapping entry (object 0xF108 (CAN Status), entry 0x05 (TX overflow))	UINT32	RO	0xF108:05, 1
1A84:13	SubIndex 019	19. PDO Mapping entry (object 0xF108 (CAN Status), entry 0x06 (Ack error))	UINT32	RO	0xF108:06, 1
1A84:14	SubIndex 020	20. PDO Mapping entry (2 bits align)	UINT32	RO	0x0000:00, 2
1A84:15	SubIndex 021	21. PDO Mapping entry (8 bits align)	UINT32	RO	0x0000:00, 8
1A84:16	SubIndex 022	22. PDO Mapping entry (16 bits align)	UINT32	RO	0x0000:00, 16

Index 1A85 TxPDO-Map CAN Status

Index (hex)	Name	Meaning	Data type	Flags	Default
1A85:0	TxPDO-Map CAN status	This PDO contains the CAN status (object 0xF108 [▶ 152]) and is optional	UINT8	RO	0x0B (11_{dec})
1A85:01	SubIndex 001	1. PDO Mapping entry (object 0xF108 (CAN Status), entry 0x01 (Bus-Off))	UINT32	RO	0xF108:01, 1
1A85:02	SubIndex 002	2. PDO Mapping entry (object 0xF108 (CAN Status), entry 0x02 (warning limit reached))	UINT32	RO	0xF108:02, 1
1A85:03	SubIndex 003	3. PDO Mapping entry (object 0xF108 (CAN Status), entry 0x03 (RX overflow))	UINT32	RO	0xF108:03, 1
1A85:04	SubIndex 004	4. PDO Mapping entry (1 bits align)	UINT32	RO	0x0000:00, 1
1A85:05	SubIndex 005	5. PDO Mapping entry (object 0xF108 (CAN Status), entry 0x05 (TX overflow))	UINT32	RO	0xF108:05, 1
1A85:06	SubIndex 006	6. PDO Mapping entry (object 0xF108 (CAN Status), entry 0x06 (Ack error))	UINT32	RO	0xF108:06, 1
1A85:07	SubIndex 007	7. PDO Mapping entry (2 bits align)	UINT32	RO	0x0000:00, 2
1A85:08	SubIndex 008	8. PDO Mapping entry (8 bits align)	UINT32	RO	0x0000:00, 8
1A85:09	SubIndex 009	9. PDO Mapping entry (16 bits align)	UINT32	RO	0x0000:00, 16
1A85:0A	SubIndex 010	10. PDO Mapping entry (object 0xF108 (CAN Status), entry 0x21 (RX error counter))	UINT32	RO	0xF108:21, 8
1A85:0B	SubIndex 011	11. PDO Mapping entry (object 0xF108 (CAN Status), entry 0x22 (TX error counter))	UINT32	RO	0xF108:22, 8

Index 1A88 TxPDO-Map CAN TxPDO Toggle 1

Index (hex)	Name	Meaning	Data type	Flags	Default
1A88:0	TxPDO-Map CAN TxPDO Toggle 1	This PDO can contain a toggle bit for each CAN TxPDO. Whether the toggle bit of a CAN TxPDO is mapped into this PDO depends on the setting in the respective TxPDO configuration object 0x8nn6 [▶ 145]. This PDO is optional	UINT8	RO	
1A88:01		first CAN TxPDO toggle bit	UINT32	RO	
...					
1A88:n		n^{th} CAN TxPDO toggle bit (if no more than 255 CAN TxPDO toggle bits are mapped, this is also the last CAN TxPDO toggle bit)	UINT32	RO	

Index 1A89 TxPDO-Map CAN TxPDO Toggle 2

Index (hex)	Name	Meaning	Data type	Flags	Default
1A89:0	TxPDO-Map CAN TxPDO Toggle 2	If more than 255 CAN TxPDO toggle bits are mapped, the additional CAN TxPDO toggle bits are contained here	UINT8	RO	0x00 (0_{dec})
1A89:01		$(n+1)^{th}$ CAN TxPDO toggle bit			
...					
1A89:m		m^{th} CAN TxPDO toggle bit			

Index 1C00 Sync manager type

Index (hex)	Name	Meaning	Data type	Flags	Default
1C00:0	Sync manager type	Using the sync managers	UINT8	RO	0x04 (4_{dec})
1C00:01	SubIndex 001	Sync-Manager Type Channel 1: Mailbox Write	UINT8	RO	0x01 (1_{dec})
1C00:02	SubIndex 002	Sync-Manager Type Channel 2: Mailbox Read	UINT8	RO	0x02 (2_{dec})
1C00:03	SubIndex 003	Sync-Manager Type Channel 3: Process Data Write (Outputs)	UINT8	RO	0x03 (3_{dec})
1C00:04	SubIndex 004	Sync-Manager Type Channel 4: Process Data Read (Inputs)	UINT8	RO	0x04 (4_{dec})

Index 1C12 RxPDO assign

Index (hex)	Name	Meaning	Data type	Flags	Default
1C12:0	RxPDO assign	PDO Assign Outputs: the RxPDOs must be assigned in the order of their indexes. The RxPDOs of the configured CANopen slaves (0x1600 [▶ 133]-0x167E) must be assigned if object 0x1C12 is transmitted in the StartUp SDOs. It can then still be decided via RxPDO Assign whether or not the RxPDO Control (index 0x1685 [▶ 134]) is transmitted in the EtherCAT output data.	UINT8	RW	
1C12:01		1. allocated RxPDO (contains the index of the associated RxPDO mapping object)	UINT16	RW	
...					
1C12:80		128. allocated RxPDO (contains the index of the associated RxPDO mapping object)	UINT16	RW	

Index 1C13 TxPDO assign

Index (hex)	Name	Meaning	Data type	Flags	Default
1C13:0	TxPDO assign	PDO Assign Inputs: the TxPDOs must be assigned in the order of their indexes. The TxPDOs of the configured CANopen slaves (0x1A00 [▶ 135]-0x1A7E) must be assigned if object 0x1C13 is transmitted in the StartUp SDOs. It can then still be decided via TxPDO Assign whether or not the TxPDOs PDO State (index 0x1A81 [▶ 135]), DiagFlag (index 0x1A82 [▶ 136]), NodeState (index 0x1A83 [▶ 136]), ExtendedDiag (index 0x1A84 [▶ 137]), CAN Status (index 0x1A85 [▶ 138]) and CAN TxPDO Toggle (index 0x1A88 [▶ 138]) are transmitted in the EtherCAT input data. In addition to the TxPDOs of the configured CANopen slaves, the TxPDOs 0x1A83 and 0x1A85 are transmitted in the default settings.	UINT8	RW	
1C13:01		1. allocated TxPDO (contains the index of the associated TxPDO mapping object)	UINT16	RW	
...					
1C13:83		135. allocated TxPDO (contains the index of the associated TxPDO mapping object)	UINT16	RW	

Index 1C32 SM output parameter

Index (hex)	Name	Meaning	Data type	Flags	Default
1C32:0	SM output parameter	Synchronisation parameters for the outputs	UINT8	RO	0x20 (32_{dec})
1C32:01	Sync mode	Current synchronisation mode: • 1: Synchronous with SM 2 event • 3: DC-Mode - Synchronous with SYNC1 event	UINT16	RW	0x0001 (1_{dec})
1C32:02	Cycle time	Cycle time (in ns): • Cycle time of the EtherCAT master	UINT32	RW	0x00000000 (0_{dec})
1C32:03	Shift time	not used	UINT32	RO	0x00000000 (0_{dec})
1C32:04	Sync modes supported	Supported synchronization modes: • Bit 1 = 1: Synchron with SM 2 event is supported • Bit 2-3 = 01: DC mode is supported • Bit 14 = 1: dynamic times (measurement through writing of 0x1C32:08 [▶ 140])	UINT16	RO	0x4006 (16390_{dec})
1C32:05	Minimum cycle time	Minimum cycle time (in ns)	UINT32	RO	0x00000000 (0_{dec})
1C32:06	Calc and copy time	Minimum time between SYNC0 and SYNC1 event (in ns, DC mode only)	UINT32	RO	0x00000000 (0_{dec})
1C32:08	Command	• 0: Measurement of the local cycle time is stopped • 1: Measurement of the local cycle time is started The entries 0x1C32:03 [▶ 140], 0x1C32:05 [▶ 140], 0x1C32:06 [▶ 140], 0x1C32:09 [▶ 140], 0x1C33:03 [▶ 141], 0x1C33:06 [▶ 140], 0x1C33:09 [▶ 141] are updated with the maximum measured values. For a subsequent measurement the measured values are reset	UINT16	RW	0x0000 (0_{dec})
1C32:09	Delay time	Time between SYNC1 event and output of the outputs (in ns, DC mode only)	UINT32	RO	0x00000000 (0_{dec})
1C32:0B	SM event missed counter	Number of missed SM events in OPERATIONAL (DC mode only)	UINT16	RO	0x0000 (0_{dec})
1C32:0C	Cycle exceeded counter	Number of occasions the cycle time was exceeded in OPERATIONAL (cycle was not completed in time or the next cycle began too early)	UINT16	RO	0x0000 (0_{dec})
1C32:0D	Shift too short counter	Number of occasions that the interval between SYNC0 and SYNC1 event was too short (DC mode only)	UINT16	RO	0x0000 (0_{dec})
1C32:20	Sync error	The synchronization was not correct in the last cycle (outputs were output too late; DC mode only)	BOOLEAN	RO	0x00 (0_{dec})

Index 1C33 SM input parameter

Index (hex)	Name	Meaning	Data type	Flags	Default
1C33:0	SM input parameter	Synchronisation parameters for the inputs	UINT8	RO	0x20 (32_{dec})
1C33:01	Sync mode	Current synchronisation mode: • 1: Synchron with SM 3 Event (no outputs available) • 3: DC - Synchron with SYNC1 Event • 34: Synchron with SM 2 Event (outputs available)	UINT16	RW	0x0022 (34_{dec})
1C33:02	Cycle time	as 0x1C32:02 [▶ 140]	UINT32	RW	0x00000000 (0_{dec})
1C33:03	Shift time	Time between SYNC0 event and reading of the inputs (in ns, only DC mode)	UINT32	RO	0x00000000 (0_{dec})
1C33:04	Sync modes supported	Supported synchronization modes: • Bit 1: Synchronous with SM 2 Event is supported (outputs available) • Bit 1: Synchronous with SM 3 Event is supported (no outputs available) • Bit 2-3 = 01: DC mode is supported • Bit 14 = 1: dynamic times (measurement through writing of 0x1C32:08 [▶ 140] or 0x1C33:08 [▶ 141])	UINT16	RO	0x4006 (16390_{dec})
1C33:05	Minimum cycle time	as 0x1C32:05 [▶ 140]	UINT32	RO	0x00000000 (0_{dec})
1C33:06	Calc and copy time	Time between reading of the inputs and availability of the inputs for the master (in ns, only DC mode)	UINT32	RO	0x00000000 (0_{dec})
1C33:08	Command	as 0x1C32:08 [▶ 140]	UINT16	RW	0x0000 (0_{dec})
1C33:09	Delay time	not supported	UINT32	RO	0x00000000 (0_{dec})
1C33:0B	SM event missed counter	as 0x1C32:11 [▶ 140]	UINT16	RO	0x0000 (0_{dec})
1C33:0C	Cycle exceeded counter	as 0x1C32:12 [▶ 140]	UINT16	RO	0x0000 (0_{dec})
1C33:0D	Shift too short counter	as 0x1C32:13 [▶ 140]	UINT16	RO	0x0000 (0_{dec})
1C33:20	Sync error	as 0x1C32:32 [▶ 140]	BOOLEAN	RO	0x00 (0_{dec})

5.5.1.3.2 Profile-specific objects (0x6000-0xFFFF)

The profile-specific objects have the same meaning for all EtherCAT slaves that support the profile 5001.

Index 6000-67E0 CAN TxPDOs Node yyy

Index (hex)	Name	Meaning	Data type	Flags	Default
6000+n*16:0	CAN TxPDOs Node yyy	This object contains the CAN TxPDOs 1-255 of the (n+1)th configured CANopen slave. The corresponding SubIndex is only present if the corresponding CAN TxPDO was also configured in object 0x8006+n*16 [▶ 145]. The object is mapped in the TxPDO (n+1) (index 0x1A00 [▶ 135]+n).	UINT8	RO	
(6000+n*16):01		Data of CAN TxPDO 1 of the (n+1)th configured CANopen slave	OCTET-STRING	RO	
...					
(6000+n*16):FF		Data of CAN TxPDO 255 of the (n+1)th configured CANopen slave	OCTET-STRING	RO	

Index 6004-67E4 CAN TxPDOs Toggle Node yyy

Index (hex)	Name	Meaning	Data type	Flags	Default
6004+n*16: 0	CAN TxPDOs Toggle Node yyy	This object contains the toggle bits of the CAN TxPDOs 1-255 of the (n+1)th configured CANopen slave. The bit toggles if the associated CAN TxPDO has been received since the previous EtherCAT input update. It does not matter whether the CAN TxPDO was received one or more times. The corresponding SubIndex is present only if the toggle bit was also configured in object 0x8006+n*16 [▶ 145]. These toggle bits are mapped in the TxPDOs 137/138 (index 0x1A88 [▶ 138] or 0x1A89 [▶ 138])	UINT8	RO	
(6004+n*16) :01		Toggle bit of CAN TxPDO 1 of the (n+1)th configured CANopen slave	BOOLEAN	RO	
...					
(6004+n*16) :FF		Toggle bit of CAN TxPDO 255 of the (n+1)th configured CANopen slave	BOOLEAN	RO	

Index 7000-77E0 CAN RxPDOs Node yyy

Index (hex)	Name	Meaning	Data type	Flags	Default
7000+n*16: 0	CAN RxPDOs Node yyy	This object contains the CAN RxPDOs 1-255 of the (n +1)th configured CANopen slave. The corresponding SubIndex is only present if the corresponding CAN Rx-PDO was also configured in object 0x8006+n*16 [▶ 145]. The object is mapped in the RxPDO (n+1) (index 0x1600 [▶ 133]+n).	UINT8	RO	
(7000+n*16) :01		Data of CAN RxPDO 1 of the (n+1)th configured CANopen slave	OCTET-STRING	RO	
...					
(7000+n*16) :FF		Data of CAN RxPDO 255 of the (n+1)th configured CANopen slave	OCTET-STRING	RO	

Index 7004-77E4 CAN TxPDOs RTR Request Node yyy

Index (hex)	Name	Meaning	Data type	Flags	Default
7004+n*16: 0	CAN TxPDOs RTR Request Node yyy	This object contains the RTR bits of the CAN TxPDOs 1-255 of the (n+1)th configured CANopen slave. If the bit is toggled, a RTR request is sent to collect the associ-ated CAN TxPDO. The corresponding SubIndex is present only if the RTR bit was also configured in object 0x8006+n*16 [▶ 145].	UINT8	RO	
(7004+n*16) :01		RTR bit of CAN TxPDO 1 of the (n+1)th configured CANopen slave	BOOLEAN	RO	
...					
(7004+n*16) :FF		RTR bit of CAN TxPDO 255 of the (n+1)th configured CANopen slave	BOOLEAN	RO	

Index 8000-87E0 Communication Parameter Node yyy

Index (hex)	Name	Meaning		Data type	Flags	Default
8000+n*16:0	Communication Parameter Node yyy	This object contains the CAN configuration of the (n+1)th configured CANopen slave (0 <= n <127). The object is to be transmitted with Complete Access, or SubIndex 0 must first be set to 0, then the individual SubIndexes transmitted (non-existent SubIndexes or gaps are thereby to be omitted) and finally SubIndex 0 set to the correct value.		UINT8	RW	0x2E (46_{dec})
(8000+n*16):01	Node address	CANopen node address of the CANopen slave, permitted values: 1-127; the entry 0xF020 [▶ 150]:(n+1) is hence automatically updated		UINT16	RW	
(8000+n*16):04	Device type	Object 0x1000 of the CANopen slave; this value is checked at the CAN boot-up, provided that the check is not disabled via the flags (SubIndex 20 of this object)		UINT32	RW	
(8000+n*16):05	Vendor ID	Object 0x1018:01 of the CANopen slave; this value is checked at the boot-up if not equal to 0		UINT32	RW	
(8000+n*16):06	Product code	Object 0x1018:02 of the CANopen slave; this value is checked at the boot-up if not equal to 0		UINT32	RW	
(8000+n*16):07	Revision	Object 0x1018:03 of the CANopen slave; this value is checked at the boot-up if not equal to 0		UINT32	RW	
(8000+n*16):08	Serial number	Object 0x1018:04 of the CANopen slave; this value is checked at the boot-up if not equal to 0		UINT32	RW	
(8000+n*16):1D	Network flags	reserved for AMS via CANopen		UINT16	RW	0x0000 (0_{dec})
(8000+n*16):1E	Network port	reserved for AMS via CANopen		UINT16	RW	0x0000 (0_{dec})
(8000+n*16):1F	Network segment address	reserved for AMS via CANopen		OCTET-STRING[6]	RW	0x00, 0x00, 0x00, 0x00, 0x00, 0x00
(8000+n*16):20	Flags	Bit 0	CAN Layer 2-Node: only asynchronous On-Change CAN PDOs are exchanged with the slave	UINT16	RW	0x0000 (0_{dec})
		Bit 1	Automatic sending of the CAN PDO communication parameters is switched off during the boot-up			
		Bit 2	reserved, must be 0			
		Bit 3	reserved, must be 0			
		Bit 4	Guarding is used instead of Heartbeat			
		Bit 5	If not all configured CAN TxPDOs have been received 10 s after the start of the CANopen slave, the CANopen slave is rebooted			
		Bit 6	The checking of object 0x1000 during the CAN boot-up is switched off			
		Bit 7	The writing of object 0x1006 during the CAN boot-up is switched off			
		Bit 8	The automatic start of the CANopen slave after completion of the CAN boot-up is switched off			
		Bit 9	reserved, must be 0			
		Bit 10	reserved, must be 0			
		Bit 11	reserved, must be 0			
		Bit 12	reserved, must be 0			
		Bit 13	reserved, must be 0			
		Bit 14	reserved, must be 0			
		Bit 15	reserved, must be 0			
(8000+n*16):21	Guarding time	Guarding time (object 0x100C or 0x1017) for Guarding or Heartbeat in accordance with bit 4 of the flags in SubIndex 0x20)		UINT16	RW	
(8000+n*16):22	Life time factor	Life time factor (object 0x100D) for Guarding or Life time factor*Guarding time (object 0x1016:01) for Heartbeat (in accordance with bit 4 of the flags in SubIndex 0x20)		UINT16	RW	
(8000+n*16):23	SDO timeout	Timeout for the transmission of CAN SDOs to the CANopen slave (in ms, 0 corresponds to 2000 ms)		UINT16	RW	0x07D0 (2000_{dec})

Index (hex)	Name	Meaning	Data type	Flags	Default
(8000+n*16):24	Boot timeout	This time is allowed to elapse after a Reset Node before the first CAN SDO is sent during the boot-up (in ms, 0 corresponds to 2000 ms)	UINT16	RW	0x07D0 (2000_{dec})
(8000+n*16):25	Parallel AoE services	Number of parallel acyclic CAN SDO orders for the CANopen slave that can be received via AoE from the EtherCAT master, saved and processed on the EL6751 (0 corresponds to the default value of 5)	UINT8	RW	0x05 (5_{dec})
(8000+n*16):26	Reaction on CANopen fault	If an error is detected during communication with the CANopen slave (error code in 0xF102 [▶ 151]:(n+ 1)), the reaction is as follows: FALSE The CANopen is stopped; the next startup (see SubIndex 0x27) will begin with Reset Node TRUE The CANopen is stopped; the next startup (see SubIndex 0x27) will begin with the first CAN StartUp SDO (usually the reading of object 0x1000)	BOOLEAN	RW	FALSE
(8000+n*16):27	Restart behavior after CANopen fault	If an error is determined during communication with the CANopen slave and the 'Reaction on CANopen fault' has been executed, the restart behavior is as follows FALSE The CANopen slave is automatically restarted (in accordance with SubIndex 0x26) TRUE The CANopen slave must be restarted via AoE	BOOLEAN	RW	FALSE
(8000+n*16):28	Master reaction after CANopen fault	If an error is determined during communication with the CANopen slave, the CANopen communication with the other CANopen slaves can be influenced: FALSE no influence TRUE a Stop Node is sent to all CANopen slaves; the CANopen communication must be restarted via AoE	BOOLEAN	RW	FALSE
(8000+n*16):29	Changes of CAN Tx-PDOs after CANopen fault	If an error is detected during communication with the CANopen slave, the EtherCAT input data is influenced as follows FALSE The data of the CAN TxPDOs in the EtherCAT input data is set to 0 TRUE The data of the CAN TxPDOs in the EtherCAT input data remains unchanged	BOOLEAN	RW	FALSE
(8000+n*16):2A		reserved for extensions; must be 0	4-bit gap	RW	0x00 (0_{dec})
(8000+n*16):2E		reserved for extensions; must be 10	UNSIGNED8	RW	0x0A (10_{dec})
(8000+n*16):2F		reserved for extensions; must be 0	8-bit gap	RW	0x00 (0_{dec})
(8000+n*16):30		reserved for extensions; must be 0	32-bit gap	RW	0x00000000 (0_{dec})
(8000+n*16):31		reserved for extensions; must be 0	32-bit gap	RW	0x00000000 (0_{dec})
(8000+n*16):32		reserved for extensions; must be 0	32-bit gap	RW	0x00000000 (0_{dec})
(8000+n*16):33		reserved for extensions; must be 0	32-bit gap	RW	0x00000000 (0_{dec})
(8000+n*16):34		reserved for extensions; must be 0	32-bit gap	RW	0x00000000 (0_{dec})
(8000+n*16):35		reserved for extensions; must be 0	32-bit gap	RW	0x00000000 (0_{dec})
(8000+n*16):36		reserved for extensions; must be 0	32-bit gap	RW	0x00000000 (0_{dec})

Index 8003-87E3 CAN SDO Init Cmds Node yyy

Index (hex)	Name	Meaning		Data type	Flags	Default
8003+n*16:0	CAN SDO Init Cmds Node yyy	This object contains the CAN StartUp SDOs of the (n+1)th configured CANopen slave (0 <= n <127), which are sent to the CANopen slave after the boot-up and before the start of the CANopen slave. Up to 255 StartUp SDOs can be configured per CANopen slave. SubIndex 0 contains the number of configured CAN StartUp SDOs. The object is to be transmitted with Complete Access, or SubIndex 0 must first be set to 0, then the individual SubIndexes transmitted and finally SubIndex 0 set to the correct value.		UINT8	RW	
(8003+n*16):01		first CAN StartUp SDO		OCTET-STRING	RW	
		Bytes 0-1	Index of the StartUp SDO			
		Byte 2	SubIndex of the StartUp SDO			
		Bytes 3-4	Length of the following data of the StartUp SDO			
		from byte 5	Data of the StartUp SDO			
...						
(8003+n*16):FF		255. CAN StartUp SDO				

Index 8006-87E6 CAN TxPDO Configuration Node yyy

Index (hex)	Name	Meaning			Data type	Flags	Default
8006+n*16:0	CAN TxPDO Configuration Node yyy	This object contains the CAN TxPDO configuration of the (n+1)th configured CANopen slave (0 <= n <127). TxPDOs 1-255 of a CANopen slave are configurable. SubIndex 0 contains the maximum configured CAN TxPDO number. If CAN TxPDOs are not present in between, the SubIndex is to be omitted or, in the case of Complete Access, filled with zeros. The object is to be transmitted with Complete Access, or SubIndex 0 must first be set to 0, then the individual SubIndexes transmitted (non-existent SubIndexes or gaps are thereby to be omitted) and finally SubIndex 0 set to the correct value.			UINT8	RW	
(8006+n*16):01		Configuration of CAN TxPDO 1 of the CANopen slave			OCTET-STRING	RW	
		Bytes 0-3	COB-ID (bits 11-31 must be 0)				
		Byte 4	Transmission Type				
		Byte 5	Length of the data of the CAN TxPDO				
		Bytes 6-7	Inhibit Time				
		Bytes 8-9	Event Time				
		Bytes 10-11	Flags				
			Bit 0	CAN TxPDO toggle (entry 0x6004 [▶ 142]+(n*16):01) is mapped into EtherCAT TxPDO 137/138 (index 0x1A88 [▶ 138]/0x1A89)			
			Bit 1-9	reserved for extensions; must be 0			
			Bit 10	Length checking is switched off			
			Bits 11-15	reserved for extensions; must be 0			
...							
(8006+n*16):FF		Configuration of CAN TxPDO 255 of the CANopen slave					

Index 8008-87E8 CAN RxPDO Configuration Node yyy

Index (hex)	Name	Meaning		Data type	Flags	Default
8008+n*16:0	CAN RxPDO Configuration Node yyy	This object contains the CAN RxPDO configuration of the (n+1)th configured CANopen slave (0 <= n <127). RxPDOs 1-255 of a CANopen slave are configurable. SubIndex 0 contains the maximum configured CAN RxPDO number. If CAN RxPDOs are not present in between, the SubIndex is to be omitted or, in the case of Complete Access, filled with zeros. The object is to be transmitted with Complete Access, or SubIndex 0 must first be set to 0, then the individual SubIndexes transmitted (non-existent SubIndexes or gaps are thereby to be omitted) and finally SubIndex 0 set to the correct value.		UINT8	RW	
(8008+n*16):01		Configuration of CAN RxPDO 1 of the CANopen slave		OCTET-STRING[12]	RW	
		Byte 0-3	COB-ID (bits 11-31 must be 0)			
		Byte 4	Transmission Type			
		Byte 5	Length of the data of the CAN RxPDO			
		Bytes 6-7	Inhibit time, is ignored by the EL6751			
		Bytes 8-9	Event Time			
		Bytes 10-11	Flags, must be 0			
...						
(8008+n*16):FF		Configuration of CAN RxPDO 255 of the CANopen slave		OCTET-STRING[12]	RW	

Index 9000-97D0 Detected CANopen Identification Node yyy

Index (hex)	Name	Meaning	Data type	Flags	Default
9000+n*16:0	Detected CANopen Identification Node yyy	This object contains the InfoData on the (n+1)th found CANopen slave, if the Scan Boxes command has been executed following switching to PREOP.	UINT8	RO	
(9000+n*16):01	Node Address	Station address of the CANopen slave (same value as in 0xF040 [▶ 150]:(n+1))	UINT16	RO	
(9000+n*16):02	Device name	Object 0x1008 of the CANopen slave	STRING	RO	
(9000+n*16):04	Device type	Object 0x1000 of the CANopen slave	UINT32	RO	
(9000+n*16):05	Vendor ID	Object 0x1018:01 of the CANopen slave	UINT32	RO	
(9000+n*16):06	Product code	Object 0x1018:02 of the CANopen slave	UINT32	RO	
(9000+n*16):07	Revision	Object 0x1018:03 of the CANopen slave	UINT32	RO	
(9000+n*16):08	Serial number	Object 0x1018:04 of the CANopen slave	UINT32	RO	

Index 9006-97D6 Detected TxPDO Configuration Node yyy

Index (hex)	Name	Meaning	Data type	Flags	Default
9006+n*16:0	Detected TxPDO Configuration Node yyy	This object contains the InfoData on the CAN TxPDOs of the (n+1)th found CANopen slave, if the Scan Boxes command has been executed following switching to PREOP.	UINT8	RO	
(9006+n*16):01		CAN TxPDO 1 (meaning of the data is identical to object 0x8yy6 [▶ 145])	OCTET-STRING[12]	RO	
...					
(9006+n*16):FF		CAN TxPDO 255	OCTET-STRING[12]	RO	

Index 9008-9085 Detected RxPDO Configuration Node yyy

Index (hex)	Name	Meaning	Data type	Flags	Default
9008+n*16:0	Detected RxPDO Configuration Node yyy	This object contains the InfoData on the CAN RxPDOs of the (n+1)th found CANopen slave, if the Scan Boxes command has been executed following switching to PREOP.	UINT8	RO	
(9008+n*16):01		CAN RxPDO 1 (meaning of the data is identical to object 0x8yy8 [▶ 146])	OCTET-STRING[12]	RO	
...					
(9008+n*16):FF		CAN RxPDO 255	OCTET-STRING[12]	RO	

Index A001-A7E1 CANopen Diagnosis Node yyy

Index (hex)	Name	Meaning		Data type	Flags	Default
A001+n*16: 0	CANopen Diagnosis Node yyy	there is a diagnostic object for each CANopen slave configured in 0x8000+n*16		UINT8	RO	
(A001+n*16):01	Flags	Bit 0	It was possible to set Producer Heartbeat; Consumer Heartbeat was rejected; despite that, the CANopen slave has been started (in order to activate monitoring on the CANopen slave, Guarding should be set instead of Heartbeat in object 0x8yy0 [▶ 143]:20)	UINT16	RO	
		Bit 1	An incorrect boot-up message was received from the CANopen slave			
		Bit 2	The CAN-Emergency-FIFO (10 emergencies can be stored) has overflowed			
		Bits 3-15	reserved for extensions			
(A001+n*16):02	Received TxPDOs	Bit 0	CAN TxPDO 1 was not received at least once after the sending of Start Node	UINT16	RO	
		Bit 1	CAN TxPDO 2 was not received at least once after the sending of Start Node			
		Bit 2	CAN TxPDO 3 was not received at least once after the sending of Start Node			
		Bit 3	CAN TxPDO 4 was not received at least once after the sending of Start Node			
		Bit 4	CAN TxPDO 5 was not received at least once after the sending of Start Node			
		Bit 5	CAN TxPDO 6 was not received at least once after the sending of Start Node			
		Bit 6	CAN TxPDO 7 was not received at least once after the sending of Start Node			
		Bit 7	CAN TxPDO 8 was not received at least once after the sending of Start Node			
		Bit 8	CAN TxPDO 9 was not received at least once after the sending of Start Node			
		Bit 9	CAN TxPDO 10 was not received at least once after the sending of Start Node			
		Bit 10	CAN TxPDO 11 was not received at least once after the sending of Start Node			
		Bit 11	CAN TxPDO 12 was not received at least once after the sending of Start Node			
		Bit 12	CAN TxPDO 13 was not received at least once after the sending of Start Node			
		Bit 13	CAN TxPDO 14 was not received at least once after the sending of Start Node			
		Bit 14	CAN TxPDO 15 was not received at least once after the sending of Start Node			
		Bit 15	all other configured CAN TxPDOs were not received at least once after the sending of Start Node			
(A001+n*16):03	CAN PDO fault	1	incorrect length of the CAN TxPDO	UINT16	RO	
		2	synchronous CAN TxPDO was not received in time			
		3	CANopen slave has automatically switched to PRE-OPERATIONAL			
		4	CAN TxPDO supervised with event time was not received in time			
		5	no response during Guarding, or failure of the Producer Heartbeat			
		6	Toggle bit has not toggled during Guarding			
		7	CANopen slave has automatically switched to STOPPED			
		8	CANopen slave sends an unknown COP state			
		9	Send queue of the EL6751 has overflowed (e.g. when no further CAN acknowledge is received during the operation)			

Index (hex)	Name	Meaning			Data type	Flags	Default
(A001+n*16):04	CAN SDO/StartUp fault	Bits 0-6	1	incorrect value when reading a StartUp SDO (details in SubIndex 7 and 8)	UINT16	RO	
			2	incorrect length when reading a StartUp SDO			
			3	SDO error when reading or writing a StartUp SDO (details in SubIndex 5 and 6)			
			4	incorrect boot-up message			
		Bit 7	0	Error during SDO upload			
			1	Error during SDO download			
		Bits 8-15	reserved for extensions				
(A001+n*16):05	Fault object (for SDO fault)	Object in which the StartUp SDO error has occurred			UINT32	RO	
(A001+n*16):06	Abort Code (for SDO fault)	Abort code of the last abort of the StartUp SDOs			UINT32	RO	
(A001+n*16):07	Read value (for SDO/ StartUp fault)	read value of the StartUp SDO			UINT32	RO	
(A001+n*16):08	Expected value (for SDO/StartUp fault)	expected value of the StartUp SDO			UINT32	RO	

Index A002-A7E2 CANopen Emergencies Node yyy

Index (hex)	Name	Meaning	Data type	Flags	Default
A002+n*16:0	CANopen Emergencies Node yyy	for each CANopen slave configured in 0x8000+n*16, there is an object that contains the received emergencies. SubIndex 0 contains the number of stored emergencies (is set to 0 if the entry 0xF103 [▶ 152]:(n+1) is set to 0)	UINT8	RO	
(A002+n*16):01		first received CAN emergency	OCTET-STRING[8]	RO	
...					
(A002+n*16):FF		last received CAN emergency	OCTET-STRING[8]	RO	

Index F000 Modular device profile

Index (hex)	Name	Meaning	Data type	Flags	Default
F000:0	Modular device profile	General information for the modular device profile	UINT8	RO	0x02 (2_{dec})
F000:01	Module index distance	Index distance of the objects of the individual channels	UINT16	RO	0x0010 (16_{dec})
F000:02	Maximum number of modules	Number of channels	UINT16	RO	0x007F (127_{dec})
F000:03	General Configuration Entries	indicates which of the SubIndexes 1-31 of the objects 0x8zz0 are supported	UINT16	RO	0x700000F9
F000:04	General Information Entries	indicates which of the SubIndexes 1-31 of the objects 0x9zz0 are supported	UINT16	RO	0x000000FD

Index F002 Detect modules command

Index (hex)	Name	Meaning		Data type	Flags	Default
F002:0	Detect modules command	The CAN bus can be scanned in PREOP for CANopen slave with this object. The CAN node addresses of the CANopen slaves found are stored in the object 0xF040 [▶ 150]. Furthermore, the InfoData objects 0x9yyz [▶ 146] are created. However, none of the objects 0x8yyz [▶ 143] or 0xF800 [▶ 154] may be transmitted before that. If this is the case, or if the scan is to be repeated, the EL6751 must be switched once to INIT and back to PREOP beforehand.		UINT8	RO	
F002:01	Command Request	The writing of this entry starts the scan; the baud rate according to 0xF800:02 [▶ 154] is located in the data word		OCTET-STRING[2]	RW	
F002:02	Command Status	0	Command ended without error, no response data	UINT8	RO	
		1	Command ended without error, response data in SubIndex 3			
		3	Command ended with an error, error code in SubIndex 3			
		100-199	0-99% of the command are ended			
		255	Command is being executed			
F002:03	Command Response	Byte 0	as SubIndex 2	OCTET-STRING[n]	RO	
		Byte 1	reserved for extensions			
		Bytes 2-3	Number of found slaves			
		Byte 4	Node address of the first CANopen slave found			
		Bytes 5-8	Vendor ID of the first CANopen slave found			
		Bytes 9-12	Product code of the first CANopen slave found			
		Byte 13	Node address of the second CANopen slave found …			
		...	etc.			

Index F020 Configured address list

Index (hex)	Name	Meaning	Data type	Flags	Default
F020:0	Configured address list	This object contains the node addresses of the configured CANopen slaves. SubIndex 0 contains the number of configured CANopen slaves. The list has a maximum of 127entries (CAN interface (if configured: node address 0 in 0xF020:01) plus 126 CANopen slaves)	UINT8	RO	
F020:01		Node address of the first configured CANopen slave (same value as in 0x8000 [▶ 143]:01)	UINT16	RO	
...					
F020:7F		Node address of the 127th configured CANopen slave (same value as in 0x87E0 [▶ 143]:01)	UINT16	RO	

Index F040 Detected address list

Index (hex)	Name	Meaning	Data type	Flags	Default
F040:0	Configured address list	This object contains the node addresses of the CANopen slaves found if the Detect modules command [▶ 150] has been executed. SubIndex 0 contains the number of CANopen slaves found. The list has a maximum of 126 entries.	UINT8	RO	
F040:01		Node address of the first CANopen slave found (same value as in 0x9000 [▶ 146]:01)	UINT16	RO	
...					
F040:7E		Node address of the 126th CANopen slave found (same value as in 0x97D0 [▶ 146]:01)	UINT16	RO	

Index F101 Extended Diag

Index (hex)	Name	Meaning	Data type	Flags	Default
F101:0	Extended Diag	This object contains the diagnosis of the EL6751, which is mapped into TxPDO 133 (Index 0x1A84 [▶ 137])	UINT8	RO	
F101:01		reserved for extensions	8-bit gap		
F101:09		reserved for extensions	3-bit gap		
F101:0C	SYNC Toggle	toggles with each transmission of the SYNC message	BOOLEAN	RO	
F101:0D	Device Diag	reserved for extensions	BOOLEAN	RO	
F101:0E	Sync Error	reserved for extensions	BOOLEAN	RO	
F101:0F	PDO Toggle	The bit toggles if the EtherCAT input data has been updated since the previous EtherCAT input update	BOOLEAN	RO	
F101:10	PDO State	This bit is set if at least one configured CANopen slave has a node state that is not equal to 0	BOOLEAN	RO	
F101:11	Cycle Counter	This counter is incremented after each CAN cycle (if at least one CANopen slave has been configured)	UINT16	RO	
F101:12	Slave Status Counter	This byte contains the number of	UINT8	RO	
F101:13		reserved for extensions	8-bit gap		
F101:14	Cycle Time	This entry contains the required cycle time of the CAN cycle in 1/9 µs	UINT16	RO	

Index F102 Node State

Index (hex)	Name	Meaning		Data type	Flags	Default
F102:0	Node State	There is a node state for each CANopen slave configured in 0x8000+n*16. The node states are mapped in TxPDO 132 (Index 0x1A83 [▶ 136]).		UINT8	RO	
F102:01		Node state of the first configured CANopen slave		UINT8	RO	
		0	No error			
		1	CANopen slave has not been restarted following an error, because entry 0x8yy0:27 has been configured for manual restart or because the CANopen slave was stopped with AoE			
		2	CANopen slave does not respond			
		4	Length of the data at a StartUp SDO upload is incorrect or StartUp SDO download has failed			
		5	Value of the data at a StartUp SDO upload is incorrect			
		8	CANopen slave is in boot-up (StartUp SDOs are being transmitted, so far no error)			
		11	CAN controller is in bus-off			
		12	CANopen slave has left OPERATIONAL (automatically or on request by AoE)			
		14	Guarding has not toggled			
		18	CANopen slave was started, all CAN TxPDOs were received, but no EtherCAT process data has been exchanged yet			
		20	CAN TxPDO with incorrect length received			
		22	synchronous or event-timer-triggered CAN TxPDO was not received in time			
		23	at least one CAN TxPDO has not yet been received after the Start Node			
		24	TX FIFO overflow (e.g. if no CAN acknowledge is detected)			
		40	CAN TxPDO with type 1 transmission was not received in this CAN cycle			
...						
F102:7F		Node state of the 127th configured CANopen slave		UINT8	RO	

Index F103 CANopen Diag Flag

Index (hex)	Name	Meaning	Data type	Flags	Default
F103:0	CANopen Diag Flag	There is a Diag Flag for each CANopen slave configured in 0x8000+n*16. The Diag Flag is set if the diagnosis (object 0xA001+((m-1)*16)) or the saved CAN emergencies (object 0xA002+((m-1)*16)) of the mth configured CANopen slave has changed. If the bit is set, the diagnosis and/or emergencies have changed. In order to reset the bit, 0 must be written to the corresponding entry (0xF103:m). The Diag Flags are mapped in TxPDO 131 (Index 0x1A82 [▶ 136]).	UINT8	RO	
F103:01		Diag Flag of the first configured CANopen slave	BOOLEAN	RW	
...					
F103:7F		Diag Flag of the last configured CANopen slave	BOOLEAN	RW	

Index F108 CAN Status

Index (hex)	Name	Meaning	Data type	Flags	Default
F108:0	CAN status	This object contains the CAN status that is mapped into TxPDOs 133 and 134 (index 0x1A84 [▶ 137] and 0x1A85 [▶ 138])	UINT8	RO	0x22 (34_{dec})
F108:01	Bus-Off	indicates whether the CAN controller reports bus-off	BOOLEAN	RO	0x00 (0_{dec})
F108:02	warning limit reached	indicates whether the CAN controller reports EWarning Limit Reached	BOOLEAN	RO	0x00 (0_{dec})
F108:03	RX overflow	RX-FIFO overflow	BOOLEAN	RO	0x00 (0_{dec})
F108:05	TX overflow	TX-FIFO overflow	BOOLEAN	RO	0x00 (0_{dec})
F108:06	Ack error	CAN acknowledge has not been detected (e.g. no CAN cable connected)	BOOLEAN	RO	0x00 (0_{dec})
F108:21	RX error counter	Rx error counter of the CAN controller	UINT8	RO	0x00 (0_{dec})
F108:22	TX error counter	Tx error counter of the CAN controller	UINT8	RO	0x00 (0_{dec})

Index F120 Diagnostic Data

Index (hex)	Name	Meaning	Data type	Flags	Default
F120:0	Diagnostic Data	This object contains additional measured times for the CAN cycle that are not contained in the Sync Manager parameter objects 0x1C32 [▶ 140]/0x1C33 [▶ 141]	UINT8	RO	
F120:01	Cycle Time	current cycle time of the CAN cycle (equivalent to 0xF101:14) in 1/9 µs	UINT32	RO	
F120:03	Maximum Cycle Time	maximum cycle time of the CAN cycle (in 1/9 µs)	UINT32	RO	
F120:04	Bus Load	CAN bus load in %	UINT16	RO	
F120:05			16-bit gap		
F120:09	Sync RxPDOs finished Time (T3)	current time after the start of the CAN cycle at which all synchronous RxPDOs were sent (in 1/9 µs)	UINT32	RO	
F120:0B	Sync RxPDOs finished Maximum Time (max T3)	maximum time after the start of the CAN cycle when all synchronous RxPDOs have been sent (in 1/9 µs)	UINT32	RO	
F120:0C	Preparing of PDOs finished Time (T2)	current time after the start of the CAN cycle at which the sending of the synchronous RxPDOs begins	UINT32	RO	
F120:0E	Preparing of PDOs finished Maximum Time (max T2)	maximum time after the start of the CAN cycle at which the sending of the synchronous RxPDOs begins	UINT32	RO	
F120:0F	Output Calc and Copy Time (T1)	current time after the start of the CAN cycle at which the SYNC message can be sent	UINT32	RO	
F120:11	Ouput Calc and Copy Maximum Time (max T1)	maximum time after the start of the CAN cycle at which the SYNC message can be sent	UINT32	RO	
F120:12	Input Calc and Copy Time (T5)	current time still required after the input shift time (0x1C33 [▶ 141]:03) until the EtherCAT input data have been completely written	UINT32	RO	
F120:14	Input Calc and Copy Maximum Time (max T5)	maximum time still required after the input shift time (0x1C33 [▶ 141]:03) until the EtherCAT input data have been completely written	UINT32	RO	
F120:15	Output Failed Counter	Number of cycles in which the EtherCAT output data were not adopted	UINT16	RO	
F120:16	Input Failed Counter	Number of cycles in which the EtherCAT input data were not collected	UINT16	RO	
F120:17	Send sync RxPDO Failed Counter	Number of CAN cycles that were omitted because the previous CAN cycle was not ended in time	UINT16	RO	
F120:18	RX Error Counter	Rx error counter (cumulative errors from 0xF108 [▶ 152]:21)	UINT16	RO	
F120:19	TX Error Counter	Tx error counter (cumulative errors from 0xF108 [▶ 152]:22)	UINT16	RO	
F120:1A		reserved for extensions	16-bit gap	RO	

Index F200 Control

Index (hex)	Name	Meaning	Data type	Flags	Default
F200:0	Control	The object contains the control data that are mapped in RxPDO 134 (index 0x1685 [▶ 134])	UINT8	RO	
F200:01	CAN Controller Auto Reset when BUS-OFF	In the case of a CAN bus-off, this allows the EL6751 to be switched again to bus-on via the process data.	BOOLEAN	RO	

Index F800 CAN Bus Parameter Set

Index (hex)	Name	Meaning		Data type	Flags	Default
F800:0	CAN Bus Parameter Set	This object contains the CAN bus parameters. The object is to be transmitted with Complete Access, or SubIndex 0 must first be set to 0, then the individual SubIndexes transmitted (non-existent SubIndexes or gaps are thereby to be omitted) and finally SubIndex 0 set to the correct value.		UINT16	RW	0x11 (17_{dec})
F800:01	Master Node Address	Node address of the CANopen master that is used for the Consumer Heartbeat		UINT8	RW	0x7F (127_{dec})
F800:02	Baud rate	0	1 Mbaud	UINT8	RW	
		1	800 kbaud			
		2	500 kbaud			
		3	250 kbaud			
		4	125 kbaud			
		5	100 kbaud			
		6	50 kbaud			
		7	20 kbaud			
		8	10 kbaud			
		255	The baud rate is determined via the bus timing register (SubIndex 5)			
F800:03	COB ID SYNC	COB ID of the SYNC message (default:0x80)		UINT16	RW	0x80 (128_{dec})
F800:04	SYNC cycle time	SYNC cycle time (must be an integer multiple of the EtherCAT cycle time (0x1C32 [▶ 140]:02))		UINT32	RW	
F800:05	Bus timing registers	byte 0	BT0 register of the SJA1000 CAN controller	UINT32	RW	0x00 (0_{dec})
		byte 1	BT1 register of the SJA1000 CAN controller			
		byte 2	must be 0			
		byte 3	must be 0			
F800:06	Slave Mode	must be 0 (CANopen master)		BOOLEAN	RW	0x00 (0_{dec})
F800:07	PDO Align 8 Bytes	0	CAN PDOs are appended to the EtherCAT process data in succession	BOOLEAN	RW	0x00 (0_{dec})
		1	each CAN PDO occupies 8 bytes in the EtherCAT process data			
F800:08		reserved for extensions		BOOLEAN	RW	0x00 (0_{dec})
F800:09		reserved for extensions		5-bit gap		0x00 (0_{dec})
F800:0E	TxPDO Delay	SYNC cycle time delay in % until the sending of the synchronous RxPDOs begins		UINT8	RW	0x1E (30_{dec})
F800:0F	CAN message queue size	Depth of the low priority CAN Tx queue (for SDOs, Heartbeat and Guarding, default: 100)		UINT16	RW	0x64 (100_{dec})
F800:10		reserved for extensions		UINT8	RW	0x00 (0_{dec})
F800:11		reserved for extensions		UINT8	RW	0x00 (0_{dec})
F800:12		reserved for extensions		16-bit gap	RW	0x00 (0_{dec})
F800:13		reserved for extensions		32-bit gap	RW	0x00 (0_{dec})
F800:14		reserved for extensions		32-bit gap	RW	0x00 (0_{dec})
F800:15		reserved for extensions		32-bit gap	RW	0x00 (0_{dec})
F800:16		reserved for extensions		32-bit gap	RW	0x00 (0_{dec})
F800:17		reserved for extensions		32-bit gap	RW	0x00 (0_{dec})
F800:18		reserved for extensions		32-bit gap	RW	0x00 (0_{dec})

5.5.2 CAN interface

5.5.2.1 CAN interface configuration

The CAN interface of the EL6751 is configured via the StartUp SDOs of the objects 0xF800, 0x8000 and 0x8001 (optional) in the PREOP state.

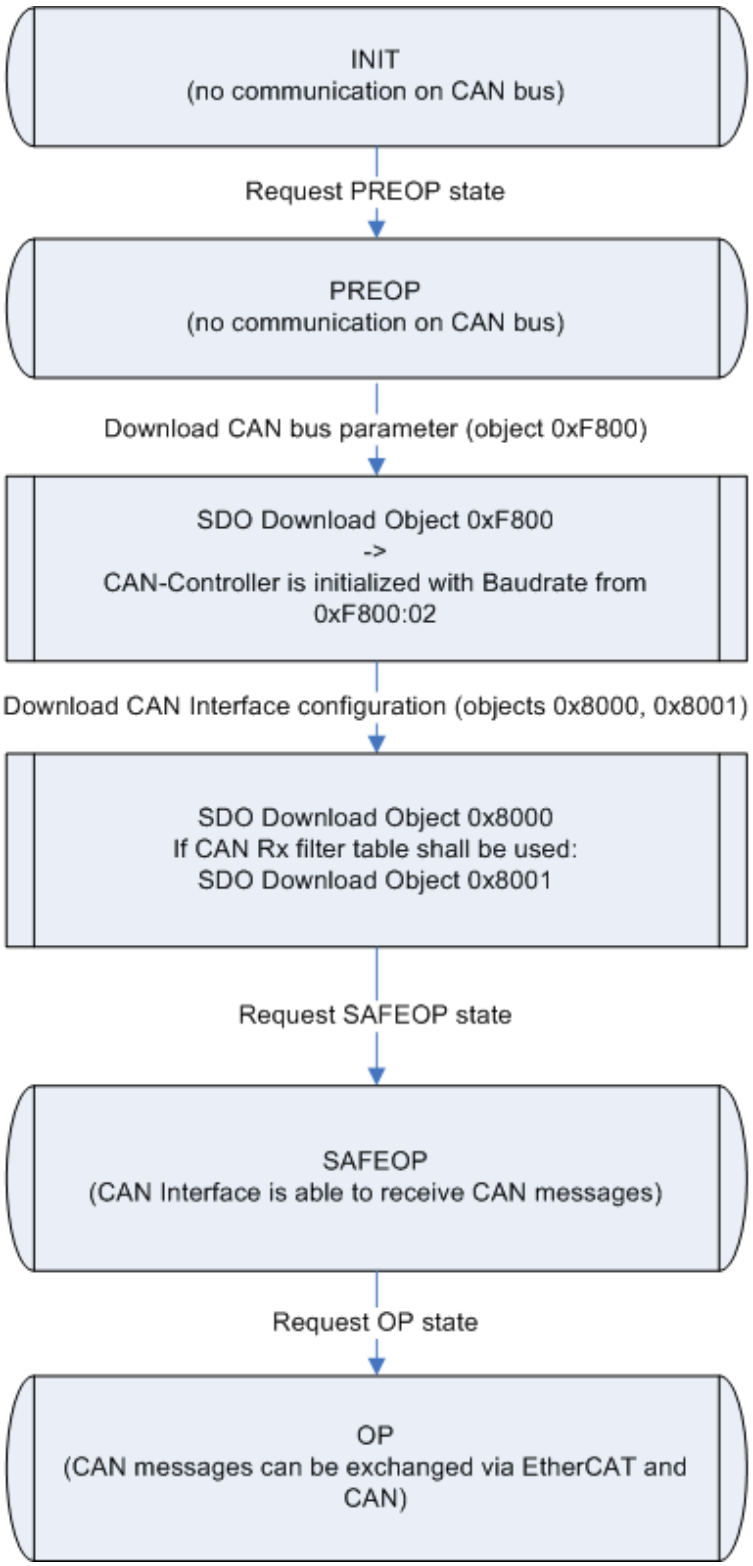


Fig. 131: Flow chart for CAN interface startup

After a power-on, the EL6751 is in the INIT state and has no CAN configuration. The CAN controller is in the OFFLINE state.

CAN bus parameters

The CANopen configuration is carried out via SDO download in the PREOP state. The objects to be loaded must be transmitted either with Complete Access or with consistency nesting (first set SubIndex 0 to 0, then write SubIndex 1-n, then set SubIndex 0 to n). Care should thereby be taken to always start with object 0xF800. After receiving the object 0xF800, the EL6751 switches the CAN controller with the appropriate baud rate from 0xF800:02 to ONLINE.

CAN interface configuration

After object 0xF800, object 0x8000 and, if Rx filter table is to be used, object 0x8001 must be transmitted.

PDO Mapping

There is one EtherCAT RxPDO and one EtherCAT TxPDO for the CAN interface. The PDO mapping of the EtherCAT PDOs is automatically calculated by the EL6751 after the download of the object 0x8000 and can be read. The PDO mapping objects can only be written with the values that the EL6751 has calculated itself. The writing of the PDO mapping thus serves only to check the PDO mapping calculated by the EtherCAT configurator and can therefore be omitted.

PDO Assign

In addition, there is one EtherCAT RxPDO and one EtherCAT TxPDO, the CAN control and CAN status. These PDOs are selected via the PDO Assign. It should thereby be observed that the EtherCAT PDOs of the CAN interface must appear in the PDO Assign. With regard to the order of the PDOs in the PDO Assign, it is important to ensure that the index of the assigned EtherCAT PDO increases with each entry in the corresponding PDO Assign object. If the EtherCAT master does not transmit any PDO Assign in the StartUp SDOs, then PDO 0x1A85 (CAN status) is transmitted alongside the CAN interface.

Cyclic communication

During the transition to SAFEOP, the EL6751 checks the length configured in the Sync Manager channels 2 and 3 against the length calculated from PDO Mapping and PDO Assign. The SAFEOP state is only adopted if these lengths match. In the SAFEOP state, the EL6751 can already receive CAN messages that are stored in the local RX queue. As soon as the EL6751 has been switched to OP, the data from the EtherCAT outputs are adopted and the CAN messages can also be exchanged via EtherCAT.

5.5.2.2 CAN interface synchronization

The CAN interface cycle, which determines the CAN messages to be sent from the EtherCAT output data and enters the CAN messages received in the EtherCAT input data, is synchronized with the EtherCAT cycle. Synchronization takes place by default via the Sync Manager 2 event. In Fast CAN Queue mode, the EL6751 can also be operated in the Distributed Clocks mode; in this case, synchronization takes place via the SYNC0 and the SYNC1 events.

Buffered CAN Queue

The following flow chart shows the sequence of the CAN cycle in the Buffered CAN Queue mode.

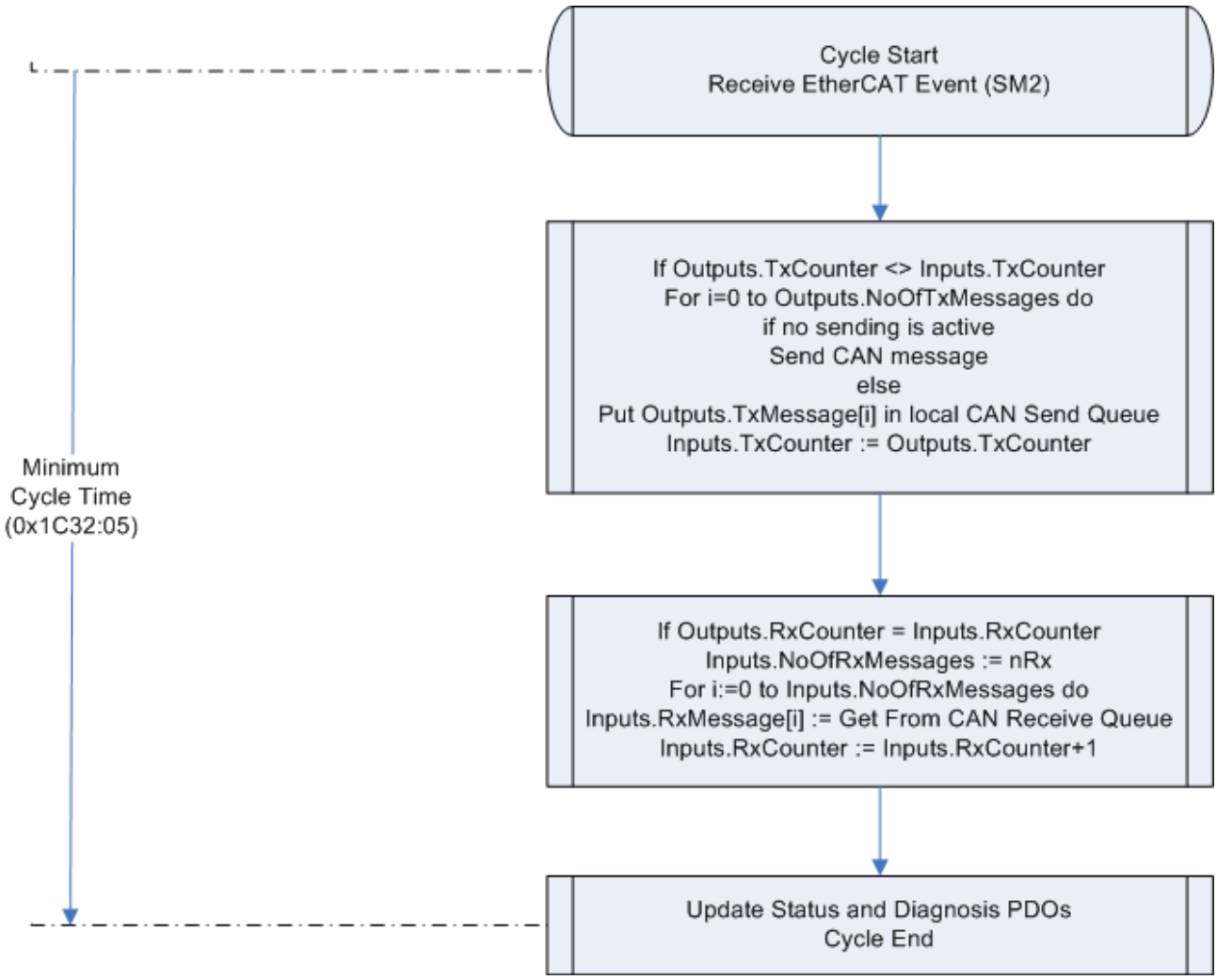


Fig. 132: Flow chart for CAN cycle in buffered CAN Queue mode

When receiving the EtherCAT process data telegram, the SM2 event is generated by the EtherCAT slave controller, thus starting the CAN interface cycle. Now it is checked whether the TxCounter (entry 0x700z:01) in the EtherCAT output data has changed. If this is the case, NoOfTxMessages (entry 0x700z:03) indicates how many CAN messages have been transmitted in the EtherCAT output data. The first CAN Tx message (entry 0x700z:04) is sent if no transmission process is active, otherwise the CAN Tx message is inserted into the local CAN send queue. The other CAN Tx messages to be sent (entries 0x700z:05 - 0x700z:03+NoOfTxMessages) are inserted into the local CAN send queue and are automatically sent as soon as the last CAN message has been sent. After that, the TxCounter in the EtherCAT input data (entry 0x600z:01) is set to the value of the TxCounter in the EtherCAT output data (0x700z:01).

Subsequently, the CAN Rx messages received since the last increment of the RxCounter in the EtherCAT input data are entered in the EtherCAT input data, provided that the RxCounter in the EtherCAT output data (entry 0x700z:02) is equal to the RxCounter in the EtherCAT input data (entry 0x600z:02). Furthermore, the number of messages entered in the EtherCAT input data (entries 0x600z :05-0x600z:03+NoOfRxMessages) is written in the NoOfRxMessages (entry 0x600z:03) of the EtherCAT input data. Then the Transaction Number (0x600z:04) of the last transmitted CAN TxMessage is entered in the EtherCAT input data and the RxCounters (entry 0x600z:02) in the EtherCAT input data are incremented.

The CAN interface cycle ends with the update of the CAN status in the EtherCAT input data.

Fast CAN Queue

The Fast CAN Queue mode essentially differs in that there is no local CAN Rx queue and that it can also be synchronized with Distributed Clocks. The CAN Rx messages received are entered directly in the EtherCAT input data; there is no longer any local storage. So that the CAN receiver always has access to EtherCAT input data, the Fast CAN Queue works only in the 3-buffer mode of the Sync Managers.

The following flow chart shows the sequence of the CAN cycle in the Fast CAN Queue mode.

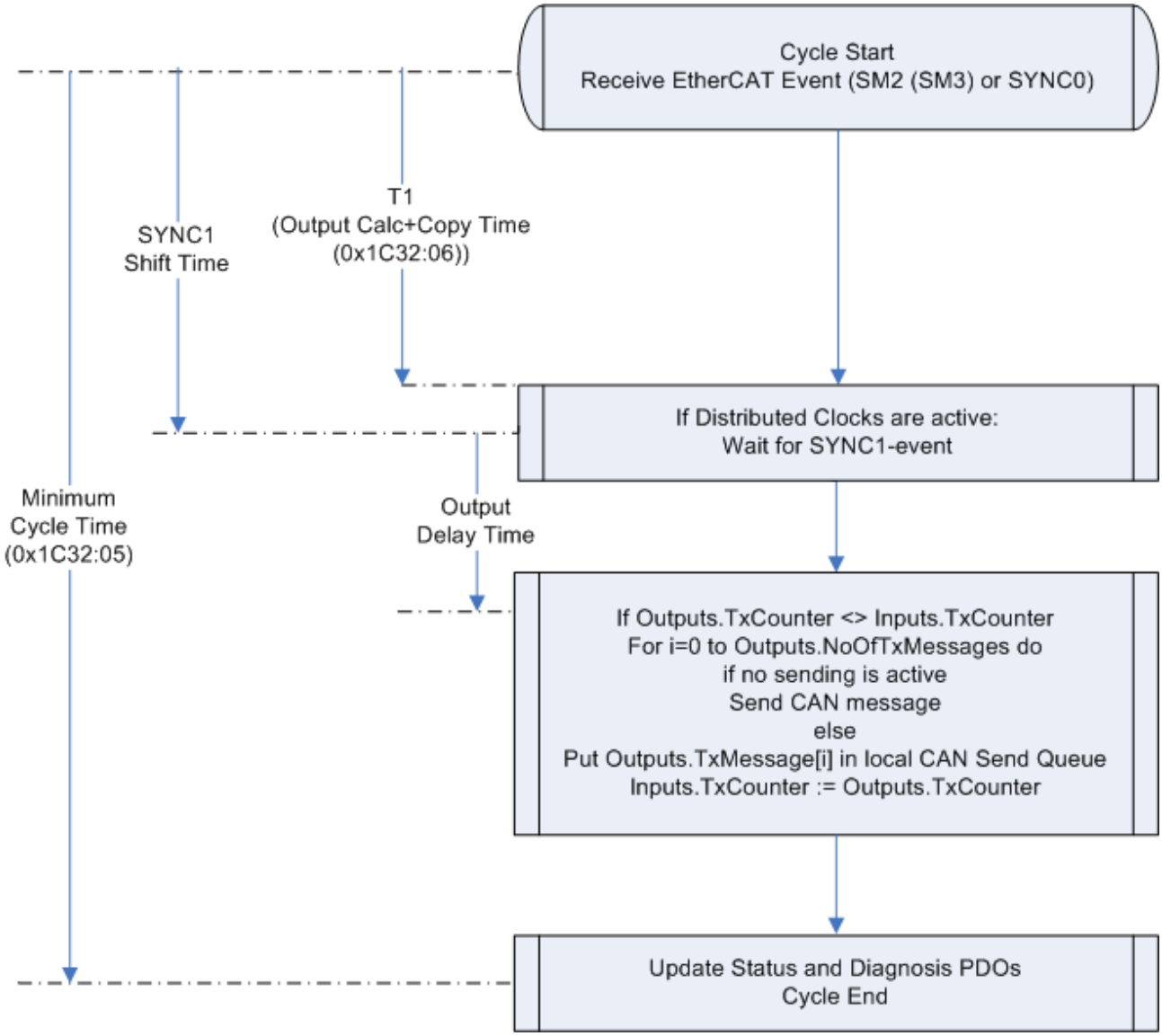


Fig. 133: Flow chart for CAN cycle in Fast CAN Queue mode

Synchronization with SM2 event

In the transmit direction, the sequence is identical to the Buffered CAN Queue mode. In the receive direction, the copying of the received CAN messages from the local Rx queue to the EtherCAT data input is dispensed with.

Synchronization with SYNC0/SYNC1 event

If distributed clocks are switched on, the CAN interface cycle would be started by the SYNC0 event. The sending of the first CAN Tx message is delayed until the SYNC1 event occurs, so that the sending of the first CAN Tx message takes place with a jitter of maximum 500 ns. The output delay time is the time between the SYNC1 event and the start of the CAN transmission of the first CAN Tx message in the CAN controller. The remaining sequence of the CAN interface cycle corresponds to that in the case of synchronization with SM2 event.

EtherCAT Update

In the EtherCAT Update it should be noted that the process data is usually transmitted with a LRW telegram. As a result of this, two cycles elapse in the task with which the EtherCAT master cycle is synchronized until the increment of the TxCounter is confirmed by the EL6751. This dead time can be avoided by selecting ‘Separate input update’ in the task, since in this case the EtherCAT output data are to be transmitted with a

LWR telegram and the EtherCAT input data just before the start of the next task cycle with a LRD telegram. A second alternative would be to allow the task (and hence the EtherCAT master) to run with half the cycle time of the CAN interface cycle.

5.5.2.3 Object description – CAN interface

If the EL6751 is used as a CAN Layer-2 interface, the following objects are available:

Index (hex)	Name
1000 [▶ 131]	Device type
1008 [▶ 131]	Device name
1009 [▶ 131]	Hardware version
100A [▶ 131]	Software version
1011 [▶ 132]	Restore default parameters
1018 [▶ 132]	Identity
10F0 [▶ 132]	Backup parameter handling
10F2 [▶ 133]	Backup parameter storage
1600 [▶ 160]	RxPDO-Map CAN Interface
1685 [▶ 134]	RxPDO-Map CAN Control
1A00 [▶ 160]	TxPDO-Map CAN Interface
1A85 [▶ 138]	TxPDO-Map CAN Status
1C00 [▶ 138]	Sync manager type
1C12 [▶ 139]	RxPDO assign
1C13 [▶ 139]	TxPDO assign
1C32 [▶ 140]	Sm output parameter
1C33 [▶ 141]	SM input parameter
6000 [▶ 161]	CAN Interface input (11-bit identifier)
6001 [▶ 161]	CAN Interface input (29-bit identifier)
7000 [▶ 161]	CAN Interface output (11-bit identifier)
7001 [▶ 161]	CAN Interface output (29-bit identifier)
8000 [▶ 162]	CAN interface configuration
8001 [▶ 162]	CAN filter table
F000 [▶ 149]	Modular device profile
F108 [▶ 131]	CAN Status
F200 [▶ 153]	CAN Control
F800 [▶ 154]	CAN bus parameter

5.5.2.3.1 Standard objects (0x1000-0x1FFF)

Here, only those objects that have a different meaning than in the CANopen master [▶ 131] are described.

Index 1600 RxPDO-Map CAN Interface

Index (hex)	Name	Meaning	Data type	Flags	Default
1600:0	RxPDO-Map CAN Interface	The CAN interface is mapped into the EtherCAT output data with this PDO. The number of buffers for the CAN messages is configured in object 0x8000. Furthermore, object 0x8000 is used to configure whether the CAN messages are transmitted with an 11-bit identifier (object 0x7000) or with a 29-bit identifier (object 0x7001). Depending on this setting, object 0x7000 or object 0x7001 is mapped in this PDO. The PDO is mandatory and must always be contained in the PDO Assign object 0x1C12	UINT8	RW	
1600:01		1. PDO Mapping entry (object 0x700z (CAN Interface output), entry 0x01 (TX Counter))	UINT32	RW	
1600:02		2. PDO Mapping entry (object 0x700z (CAN Interface output), entry 0x02 (RX Counter))	UINT32	RW	
1600:03		3. PDO Mapping entry (object 0x700z (CAN Interface output), entry 0x03 (Number of TX Messages))	UINT32	RW	
1600:04		4. PDO Mapping entry (object 0x700z (CAN Interface output), entry 0x04 (TX Message 1))	UINT32	RW	
...		..			
1600:m		m. PDO Mapping entry (object 0x700z (CAN Interface output), entry m (TX Message m-3))	UINT32	RW	

Index 1A00 TxPDO-Map CAN Interface

Index (hex)	Name	Meaning	Data type	Flags	Default
1A00:0	TxPDO-Map CAN Interface	The CAN interface is mapped into the EtherCAT input data with this PDO. The number of buffers for the CAN messages is configured in object 0x8000. Furthermore, object 0x8000 is used to configure whether the CAN messages are transmitted with an 11-bit identifier (object 0x6000) or with a 29-bit identifier (object 0x6001). Depending on this setting, object 0x6000 or object 0x6001is mapped in this PDO. The PDO is mandatory and must always be contained in the PDO Assign object 0x1C13	UINT8	RW	
1A00:01		1. PDO Mapping entry (object 0x6000 (CAN Interface input), entry 0x01 (TX Counter))	UINT32	RW	
1A00:02		2. PDO Mapping entry (object 0x6000 (CAN Interface input), entry 0x02 (RX Counter))	UINT32	RW	
1A00:03		3. PDO Mapping entry (object 0x6000 (CAN Interface input), entry 0x03 (Number of RX Messages))	UINT32	RW	
1A00:04		4. PDO Mapping entry (object 0x6000 (CAN Interface input), entry 0x04 (TX Transaction Number))	UINT32	RW	
1A00:05		5. PDO Mapping entry (object 0x6000 (CAN Interface input), entry 0x05 (RX Message 1))	UINT32	RW	
...		..			
1A00:m		m. PDO Mapping entry (object 0x6000 (CAN Interface input), entry m (RX Message m-4))	UINT32	RW	

5.5.2.3.2 Profile-specific objects (0x6000-0xFFFF)

The profile-specific objects have the same meaning for all EtherCAT slaves that support the profile 5001.

Index 6000 CAN Rx message queue

Index (hex)	Name	Meaning	Data type	Flags	Default
6000:0	CAN Rx message queue	This object contains the inputs of the CAN interface with 11-bit identifier.	UINT8	RO	
6000:01	TX counter	see CAN interface description	UINT16	RO	
6000:02	RX counter	see CAN interface description	UINT16	RO	
6000:03	Number of RX Messages	see CAN interface description	UINT16	RO	
6000:04	TX Transaction Number	see CAN interface description	UNT16	RO	
6000:05	RX Message 1	see CAN interface description	OCTET-STRING[10]	RO	
...					
6000:m	RX Message m-4	see CAN interface description	OCTET-STRING[10]	RO	

Index 6001 CAN Rx extended message queue

Index (hex)	Name	Meaning	Data type	Flags	Default
6001:0	CAN Rx extended message queue	This object contains the inputs of the CAN interface with 29-bit identifier.	UINT8	RO	
6001:01	TX counter	see CAN interface description	UINT16	RO	
6001:02	RX counter	see CAN interface description	UINT16	RO	
6001:03	Number of RX Messages	see CAN interface description	UINT16	RO	
6001:04	TX Transaction Number	see CAN interface description	UNT16	RO	
6001:05	RX Message 1	see CAN interface description	OCTET-STRING[14]	RO	
...					
6001:m	RX Message m-4	see CAN interface description	OCTET-STRING[14]	RO	

Index 7000 CAN Tx message queue

Index (hex)	Name	Meaning	Data type	Flags	Default
7000:0	CAN Tx message queue	This object contains the outputs of the CAN interface with 11-bit identifier.	UINT8	RO	
7000:01	TX counter	see CAN interface description	UINT16	RO	
7000:02	RX counter	see CAN interface description	UINT16	RO	
7000:03	Number of TX Messages	see CAN interface description	UINT16	RO	
7000:04	TX Message 1	see CAN interface description	OCTET-STRING[12]	RO	
...					
7000:m	TX Message m-3	see CAN interface description	OCTET-STRING[12]	RO	

Index 7001 CAN Tx extended message queue

Index (hex)	Name	Meaning	Data type	Flags	Default
7001:0	CAN Tx extended message queue	This object contains the outputs of the CAN interface with 11-bit identifier.	UINT8	RO	
7001:01	TX counter	see CAN interface description	UINT16	RO	
7001:02	RX counter	see CAN interface description	UINT16	RO	
7001:03	Number of TX Messages	see CAN interface description	UINT16	RO	
7001:04	TX Message 1	see CAN interface description	OCTET-STRING[16]	RO	
...					
7001:m	TX Message m-3	see CAN interface description	OCTET-STRING[16]	RO	

Index 8000 CAN Interface configuration

Index (hex)	Name	Meaning		Data type	Flags	Default
8000:0	CAN interface configuration	the CAN interface is configured with this object		UINT8	RO	0x24 (36_{dec})
8000:01	Node address	must be set to 0		UINT16	RW	0x0000 (0_{dec})
8000:20	Flags	Bits 0-2	reserved for extensions, must be 0	UINT16	RW	0x0000 (0_{dec})
		Bit 3	0 = Standard Queue (11 Bit Identifier), 1 = Extended Queue (29 Bit Identifier)			
		Bits 4-8	reserved for extensions, must be 0			
		Bit 9	0 = Buffered CAN Queue, 1 = Fast CAN Queue (non buffered)			
		Bits 10-15	reserved for extensions, must be 0			
8000:21	Rx queue size	Number of RX messages		UINT8	RW	
8000:22	Tx queue size	Number of TX messages		UINT8	RW	
8000:23		reserved for extensions; must be 150		UINT16	RW	0x0096 (150_{dec})
8000:24		reserved for extensions; must be 150		UINT16	RW	0x0096 (150_{dec})

Index 8001 CAN Rx filter table

From firmware 17 of the EL6751, parameter 0x8001 must be written with valid values.

If all data are to be written into the CAN interface, the following must be entered:

For 11 bit and 29 bit identifiers:
0x8001: 01 00 00 00 00 00 00 FF FF FF 1F

For 11 bit identifiers
0x8001: 01 00 00 00 00 00 00 00 FF 07 00 00

Index (hex)	Name	Meaning	Data type	Flags	Default
8001:0	CAN Rx filter table	Number of valid filter sub index values (1..m, m = 255). With this object, the CAN identifier ranges can be defined for the identifiers CAN messages, which are entered in the Rx queue and transferred with the EtherCAT input data. This object must be configured from firmware 17.	UINT8	RO	
8001:01	Identifier Area 1	Bytes 0-3: first identifier that is entered in the Rx queue	UINT64	RO	
		Bytes 4-7: last identifier that is entered in the Rx queue			
...					
8001:m	Identifier Area m	Bytes 0-3: first identifier that is entered in the Rx queue	UINT64	RO	
		Bytes 4-7: last identifier that is entered in the Rx queue			

6 Error handling and diagnostics

6.1 EL6751 – LED description

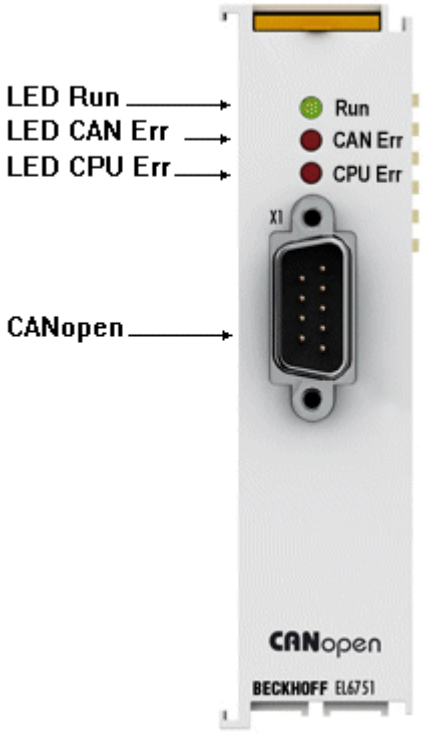


Fig. 134: LEDs

LED behavior

The most important states of the terminal can be quickly diagnosed on the basis of the LEDs:

LED	Color	Meaning	
RUN	green	This LED indicates the terminal's operating state:	
		off	State of the EtherCAT State Machine: **INIT** = initialization of the terminal; **BOOTSTRAP** = function for terminal firmware updates
		Flashes at 2 Hz	State of the EtherCAT State Machine: **PREOP** = function for mailbox communication and different standard-settings set
		Flashes at 1 Hz	State of the EtherCAT State Machine: **SAFEOP** = verification of the sync manager channels and the distributed clocks. Outputs remain in safe state
		on	State of the EtherCAT State Machine: **OP** = normal operating state; mailbox and process data communication is possible
Err	red	off	All configured bus devices are error-free (box state = 0); TwinCAT task or process is running.
		Flashes at 1 Hz	At least one box state is not equal to 0 (e.g. device not found, wrong configuration, device in error state)
		Flashes at 10 Hz	Configuration upload being carried out
		on	CAN controller is BUS OFF. Physical CAN problem. Possible causes: e.g. missing termination resistor, bus line too long, wrong baud rate, node address assigned twice, wiring error, short-circuit. Restart required
CPU error	red	on	EL6751 processor error
		Flashes at 10 Hz	The EL6751 processor starts

6.2 EL6751 – Bus node diagnostics

The CANopen fieldbus card EL6751 has a comprehensive range of diagnostic options for connected network nodes.

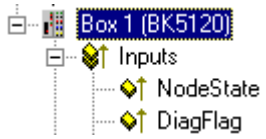


Fig. 135: Diagnosis of inputs in the TwinCAT tree

For each CANopen fieldbus node there is a node state input variable, which signals the status of the current slave during the runtime and can be linked, for example with the PLC.

Node State (Box-State)

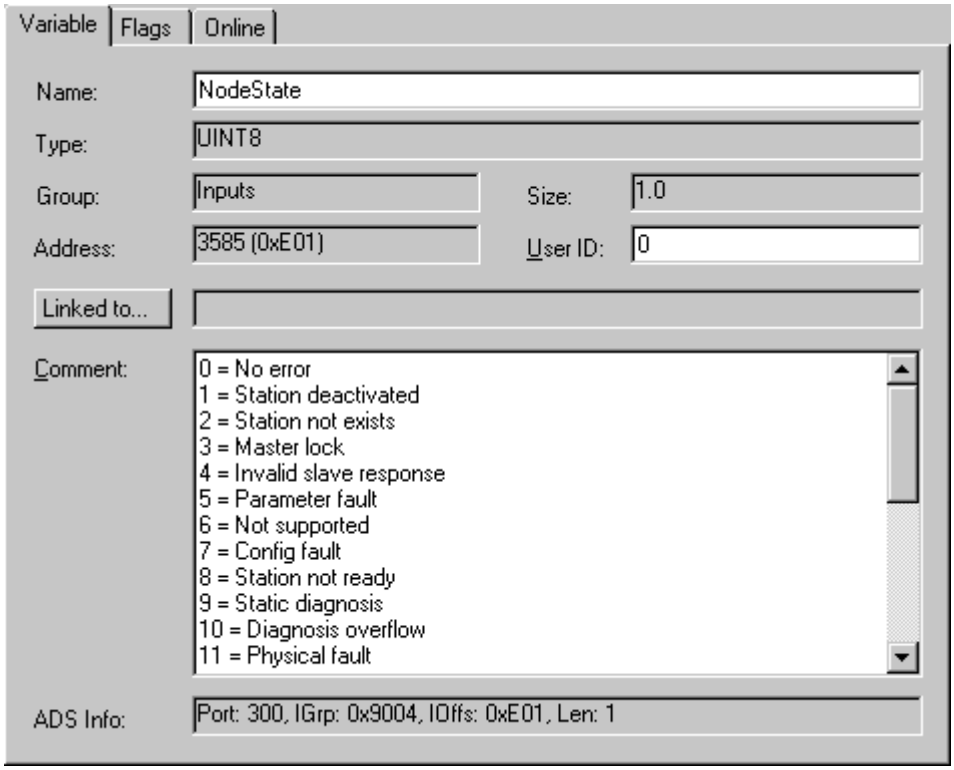


Fig. 136: "Variable" tab

Node State	Meaning	Explanation
0 = 0x00	No error	Bus node is operational, communication is running correctly
1 = 0x01	Node deactivated	The node is subject to one or more of the following errors: • guarding/heartbeat error (failure, toggle bit error, node has changed state) • expected TxPDO has not been received • TxPDO length shorter than expected Node has been stopped, because "Manual restart [▶ 89]" following a node failure has been selected.
2 = 0x02	Node not found	Node not found: no answer to SDO read access to object 0x1000 at the expected node address. Check the following at the node: what node address is set, and what baud rate. Check network (terminating resistors, connectors, bus length, crossed wiring etc.)
4 = 0x04	SDO syntax error at StartUp	Error during SDO write access: SDO abort by node. See the "Diag" tab for details. or: the length of an object read by SDO does not agree with the expected length.
5 = 0x05	SDO data mismatch at StartUp	Expected data does not agree with that read via SDO (e.g. device profile and/or additional info do not agree with object 0x1000). Can also occur if the value to be written (e.g. PDO COB-ID) is read back due to refusal of write access, and does not agree. See the "Diag" tab for details.
8 = 0x08	Node StartUp in progress	Node was found and has been started.
11 = 0x0B	EL6751Bus-OFF	CAN chip has entered the "Bus-OFF" state: transmit error counter is running
12 = 0x0C	Pre-Operational	Node has gone pre-operational (on its own account).
13 = 0x0D	Severe bus fault	General firmware error
14 = 0x0E	Guarding: toggle error	Guarding error: Toggle bit has not changed
20 = 0x14	TxPDO too short	Received TxPDO shorter than expected
22 = 0x16	Expected TxPDO is missing	• TxPDO has not been received within the expected time interval: • sync interval with synchronous TxPDOs, • event timer with event-driven PDOs)
23 = 0x17	Node is Operational but not all TxPDOs were received	Node has been started, but at least one TxPDO has not yet been received from the node. Possible causes (e.g.): • The node only sends event-driven PDOs after the first event (this is not the intention of the CANopen specification, but is quite usual) • Too many TxPDOs have been configured • A TxPDO is present at the node, but no process data has been mapped • The TxPDO has transmission type 1...120 (synchronous), but SYNC has not yet been sent because the associated task has not been started

DiagFlag

Shows whether the box diagnostic information has changed.

Reading the Diagnostic Data via ADS

CANopen emergencies and other diagnostic data can be read out via ADS read (new data present as soon as you see the DiagFlag). The ADS Net-ID of the EL6751 must be entered for this. Other ADS parameters:

Port: 200

IndexGroup: Lo-Word = 0xF180, Hi-Word = Node-Number.

IndexOffset: See below

Length: See below

If more than 26 bytes of diagnostic data have been read out the emergency memory is reset. The DiagFlag is reset as soon as at least 108 bytes have been read starting from offset 0. Alternatively, the flag is reset after each of read access, if IndexGroup 0xF181 (instead of 0xF180) is used for the read.

The diagnostic data have the following definitions:

Offset 0,1:	Bit 1:	Boot up message not received or incorrect
	Bit 2:	Emergency-Overflow
	Bit 0, Bit 3-15:	reserved
Offset 2,3:	Bit 0-14:	TX-PDO (i+1) received
	Bit 15:	All TX PDOs 16-n received
Offset 4,5:	Bit 0-4:	1: Incorrect TX PDO length
		2: Synchronous TX PDO absent
		3: Node signaling PRE-OPERATIONAL
		4: Event timer timed out for TX PDO
		5: No response and guarding is activated
		6: Toggling missed several times and guarding activated
	Bit 5-15:	Associated COB ID
Offset 6:	Bit 0-7:	1: Incorrect value during SDO upload
		2: Incorrect length during SDO upload
		3: Abort during SDO up/download
		4: Incorrect date during a boot-up message
		5: Timeout while waiting for a boot-up message
Offset 7:	Bit 0-7:	2: Incorrect SDO command specifier
		3: SDO toggle bit has not changed
		4: SDO length too great
		5: SDO-Abort
		6: SDO-Timeout
Offset 8,9	Bit 0-7:	SDO up/download index
Offset 10:	Bit 0-7:	SDO up/download sub-index
Offset 11:	Bit 0-7:	reserved
Offset 12:	Bit 0-7:	Abort errorClass
Offset 13:	Bit 0-7:	Abort errorCode
Offset 14,15:	Bit 0-15:	Abort additionalCode
Offset 16-19:		Read value (if offset 6 = 1)
Offset 20-23:		Expected value (if offset 6 = 1)
Offset 24-25:		Number of consecutive emergencies
Offset 26 - n:		Emergencies (8 bytes each)

6.3 EL6751 diagnostics

Diagnostic Inputs

The EL6751 has various diagnostic variables that describe the state of the terminal and the CANopen network:

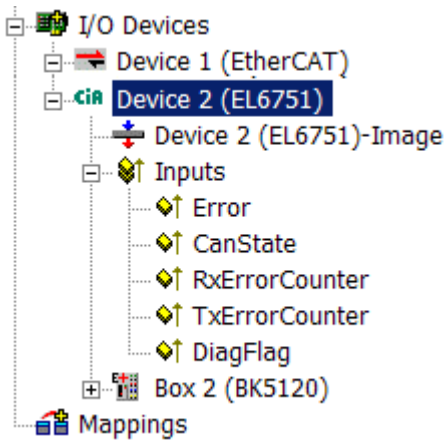


Fig. 137: TwinCAT tree: Diagnostic variables of the EL6751

Error
Shows the number of slaves whose Box State is not equal to zero. Only check the BoxState of the slaves if this value is other than 0.

CANState

Bit 0: CAN controller is in BUS OFF state; due to an excessive number of CAN errors (error frames) the CAN controller no longer takes part in the bus traffic; in this case there is a serious physical error in the CAN network (e.g. insufficient or too many termination resistors, at least one device with invalid baud rate, short circuit, etc.). The Bus Off state can only be quit with a CAN node reset.
Bit 1: CAN controller warning limit reached; the send or receive error counter of the CAN controller has exceeded 96.
Bit 2: Rx queue overrun; overflow of the internal receive buffer. Data retrieval by the controller is too slow.
Bit 3: Hi-Prio Tx queue overrun; transmit buffer overflow for PDOs and SYNC messages.
Bit 4: Lo-Prio Tx queue overrun; transmit buffer overflow for SDOs, guarding, heartbeat, etc.
Bit 5: CAN transmission error; this bit is set when no data can be transmitted, e.g. if the EL6751 connector is unplugged.
Bit 7: internal Rx queue full; the data are not read in via the CAN interface (function available from firmware 9).
Bit 15: toggles when the CAN-SYNC message is sent. This enables the function of the CAN multiplier (CAN transmission in every n^{th} EtherCAT cycle) to be checked.

RxErrorCounter
Faulty receive data; this is set to a certain value in the event of an error and decremented to one once faultless communication has resumed.

TxErrorCounter
Faulty transmission data; this is set to a certain value in the event of an error and decremented to one once faultless communication has resumed.

DiagFlag: Shows whether the diagnostics information on the card has changed. This can be read off using ADS-Read. For that purpose, specify the net ID of the EL6751, the port number 200 and the IndexGroup 0xF100. The IndexOffset and the length then relate to the diagnostic data. (Note: The Box States are also available as box variables.)

Offset 1-127: BusStatus List, 1-127 one byte each station address which contains the station status (see BoxState for CANopen boxes)

6.4 EL6751- Emergency messages

The EL6751 stores incoming emergency messages in the diagnostic area from offset 26 (see below). Up to 10 emergencies can be stored for each bus node. The oldest message is replaced if more emergencies than this arrive.

New diagnostic data (emergencies or other diagnostic data) is present as soon as the DiagFlag is set.

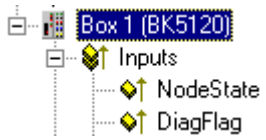


Fig. 138: TwinCAT tree: Diagnostic Inputs

CANopen emergencies and other diagnostic data can be read via ADS. You need to enter the EL6751 ADS net ID. Other ADS parameters:

Port: 200

IndexGroup: Lo-Word = 0xF180, Hi-Word = Node-Number.

IndexOffset: See below

Length: See below

If more than 26 bytes of diagnostic data have been read out the emergency memory is reset. The DiagFlag is reset as soon as at least 108 bytes have been read starting from offset 0. Alternatively, the flag is reset after each of read access, if IndexGroup 0xF181 (instead of 0xF180) is used for the read.

A description of the diagnostic data at offset 0...23 is to be found in the corresponding Section [▶ 164]. The diagnostic area starting at offset 24 is organized as follows:

Offset 24-25: Number of consecutive emergencies

Offset 26 - n: Emergencies (8 bytes each)

The significance of the emergency data is to be found in the technical documentation for the particular CANopen device.

6.5 EL6751 - ADS Error Codes

The ADS error codes have the following meaning:

Error	Description
0x1001	Insufficient memory for AMS command
0x1101	Incorrect data length at StartFieldbus
0x1102	Incorrect DeviceState at StartFieldbus
0x1103	Device cannot change from INIT to RUN
0x1104	Incorrect AdsState in INIT state
0x1105	Incorrect DeviceState at StopFieldbus
0x1106	Device cannot change from STOP to RUN if a CDL is not defined
0x1107	Device cannot change from STOP to RUN if a box is not defined
0x1108	Incorrect data length at StartDataTransfer
0x1109	Incorrect DeviceState at StartDataTransfer
0x110A	Incorrect AdsState in STOP state
0x110B	Device cannot change from RUN to INIT
0x110C	Incorrect data length at StopDataTransfer
0x110D	Incorrect DeviceState at StopDataTransfer
0x1110	Incorrect AdsState in RUN state
0x1111	Loading the device parameters is only permitted in the INIT state
0x1112	Incorrect data length at SetDeviceState
0x1113	AddBox not allowed in INIT state
0x1114	Incorrect data length at AddBox
0x1115	DeleteBox not allowed in INIT state
0x1116	Incorrect IndexOffset at DeleteBox
0x1117	Incorrect data length at DeleteBox
0x1118	ReadBox only with AdsRead
0x1119	AddCdl not allowed in INIT state
0x111A	Incorrect data length at AddCdl
0x111B	DeleteCdl not allowed in INIT state
0x111C	Incorrect IndexOffset at DeleteCdl
0x111D	Incorrect data length at DeleteCdl
0x111E	Incorrect IndexGroup at AdsWrite
0x111F	Device parameters cannot be read
0x1120	Box parameters cannot be read
0x1121	Cdl parameters cannot be read
0x1122	DeleteBox or DeleteCdl only with AdsWrite
0x1123	ReadBox only possible in STOP state
0x1124	Incorrect IndexOffset at ReadBox
0x1125	Incorrect data length at ReadBox
0x1126	Incorrect IndexGroup at AdsRead
0x1127	AddDeviceNotification not allowed in INIT state
0x1128	DelDeviceNotification not allowed in INIT state
0x1129	IndexOffset too large during reading of the device diagnostic data
0x112B	IndexOffset too large during reading of the box diagnostic data
0x112F	Insufficient memory for ReadBox response
0x1201	AddCdl: CDL no. is too large
0x1202	DeleteCdl only possible when CDL is stopped
0x1203	DeleteCdl not possible as no CDL defined
0x1204	Cycle could not be completed within the internal watchdog time

Error	Description
0x1301	AddCdl: I/O access multiplier is too large
0x1302	AddCdl: Start cycle must be smaller than I/O access multiplier
0x1303	AddCdl: Incorrect data length for output area
0x1304	AddCdl: Incorrect data offset for output area
0x1305	AddCdl: Output area is already defined
0x1306	AddCdl: Incorrect data length for input area
0x1307	AddCdl: Incorrect data offset for input area
0x1308	AddCdl: Input area is already defined
0x1309	AddCdl: Incorrect area type
0x130A	AddCdl: BoxNo has not been defined with AddBox
0x130B	AddCdl: Incorrect action type
0x130C	AddCdl: Insufficient memory for poll list
0x130D	AddCdl: Insufficient memory for poll list array
0x130E	AddCdl: Insufficient memory for actions
0x130F	AddCdl: CdlNo already exists
0x1310	DeleteCdl: CDL is not stopped
0x1311	AddCdl: Insufficient memory for asynchronous transmit list
0x1312	AddCdl: Insufficient memory for synchronous receive list
0x1313	AddCdl: Insufficient memory for asynchronous receive list
0x1316	AddCdl: Insufficient memory for synchronous receive list
0x1318	AddCdl: Only slave action allowed
0x1319	AddCdl: Insufficient memory for slave list
0x1601	AddBox: BoxNo is too large
0x1602	AddBox: Insufficient memory for ADS StartUp telegram
0x1604	DeleteBox: Box is not stopped
0x1605	AddBox: Insufficient memory for CDL telegram
0x1606	AddBox: Number of CDL telegrams is too large
0x1607	BoxRestart: Box is not stopped
0x1608	BoxRestart: AdsWriteControl syntax error
0x1609	BoxRestart: Incorrect AdsState
0x160A	Syntax error in AdsWrite to box port
0x160B	AMS CmdId is not supported by box port
0x160E	AdsReadState is not supported by box port
0x160F	AddBox: Insufficient memory for the ADS interface
0x1610	AddBox: AMS channel is invalid
0x1611	Error communicating with an AMS box
0x1613	Error communicating with an AMS box: Incorrect offset
0x1614	Error communicating with an AMS box: Data packet is too large
0x1615	Error communicating with an AMS box: AMS command is too large
0x1616	Error communicating with an AMS box: First data packet is too large
0x1617	Error communicating with an AMS box: First offset is incorrect
0x1701	AddDeviceNotification: Length of device diagnostic data to small
0x1702	AddDeviceNotification: Length of device diagnostic data to large
0x1703	AddDeviceNotification: Length of box diagnostic data to small
0x1704	AddDeviceNotification: Length of box diagnostic data to large
0x1705	AddDeviceNotification: Box is not defined
0x1706	AddDeviceNotification: Incorrect IndexGroup
0x1707	AddDeviceNotification: No more resources for client
0x1708	DelDeviceNotification: Incorrect handle

Error	Description
0x1801	StartFieldbus: In equidistant operation, shift time + safety time + 2*PLL sync. time must be greater than the cycle time
0x1802	StartFieldbus: Cycle time is too large
0x1803	StartFieldbus: Cycle time is too large
0x1804	StartFieldbus: Shift time is too large
0x1805	StartFieldbus: PLL sync time is too large
0x1806	StartFieldbus: Safety time is too large
0x1807	StartFieldbus: Cycle times shorter than 1 ms must be integral divisors of 1 ms

Error	Description
0x1A01	Memory could not be allocated from the huge heap, because it is larger than 0x8000 bytes
0x1A02	Memory could not be allocated from the near heap, because it is larger than 0x1000 bytes
0x1A03	Memory could not be allocated from the huge heap, because it is 0 bytes
0x1A04	Memory could not be allocated from the near heap, because it is 0 bytes
0x2001	StartFieldbus: Initialization of the CAN controller failed
0x2002	AddBox: Incorrect box parameter length
0x2003	AddBox: Incorrect box number
0x2004	AddBox: Syntax error in ADS StartUp parameters
0x2005	AddBox: Syntax errors in PDO parameters
0x2006	AddBox: Syntax error in data length
0x2007	AddBox: Insufficient memory
0x2008	AddCdl: Incorrect receive data length
0x2009	AddCdl: Incorrect transmit data length
0x200A	AddCdl: PDO is not defined
0x200B	AddCdl: PDO Id is already defined
0x200C	AddBox: Syntax error in ADS StartUp parameters
0x200D	AddBox: Syntax error in ADS StartUp parameters
0x200E	AddBox: Emergency Id is already defined
0x200F	AddBox: Too many PDOs defined
0x2010	AddCdl: Incorrect telegram index
0x2011	AddBox: Too many Rx or Tx PDOs
0x2012	AdsRead: Incorrect IndexGroup
0x2013	AdsRead: Incorrect IndexOffset
0x2014	AdsRead: Incorrect length
0x2015	AdsWrite: Incorrect IndexGroup
0x2016	AdsWrite: Incorrect IndexOffset
0x2017	AdsWrite: Incorrect length
0x2018	AddBox: Guarding time smaller than 10 is not possible
0x2019	AddBox: Incorrect transmission type in CAN Layer 2 node
0x201A	AdsRead: not possible at CAN Layer 2 node
0x201B	AdsWrite: not possible at CAN Layer 2 node
0x201C	AddBox: BootUp Id is already defined
0x201D	AddBox: BoxNo 0 is not possible
0x201E	StartFieldbus: Loading the device parameters is only possible in the OFFLINE state
0x201F	StartDataTransfer: No memory for copy queue
0x2020	ReadBox: no more memory
0x2021	ReadBox: SDO error or timeout
0x2022	ReadBox: SDO cannot be initialized
0x2023	StartFieldbus: reserved device parameter not equal to 0
0x2101	Insufficient memory for low-priority queues
0x2102	Insufficient memory for low-priority queues
0x2103	Insufficient memory at node boot-up
0x2104	Insufficient memory at node boot-up
0x2105	Insufficient memory at node boot-up
0x2106	Insufficient memory at node boot-up
0x2107	Insufficient memory at node boot-up
0x2108	Insufficient memory at node boot-up
0x2109	Insufficient memory at node boot-up

Error	Description
0x210A	Insufficient memory at node boot-up
0x210B	Insufficient memory at node boot-up
0x210C	Insufficient memory at node boot-up
0x210D	Insufficient memory at node boot-up
0x210E	Insufficient memory at node boot-up
0x210F	Insufficient memory at node boot-up
0x2110	Insufficient memory at node boot-up
0x2111	Insufficient memory at node boot-up
0x2112	Insufficient memory at node boot-up
0x2113	Insufficient memory at node boot-up
0x2114	Insufficient memory at node boot-up
0x2301	Insufficient memory for low-priority queues
0x2302	Insufficient memory for low-priority queues

6.6 CANopen Trouble Shooting

Error Frames

One sign of errors in the CAN wiring, the address assignment or the setting of the baud rate is an increased number of error frames: the diagnostic LEDs then show *Warning Limit exceeded* or *Bus-off state entered*.

Error Frames

Warning limit exceeded, passive error or bus-off state are indicated first of all at those nodes that have detected the most errors. These nodes are not necessarily the cause for the occurrence of error frames!
If, for instance, one node contributes unusually heavily to the bus traffic (e.g. because it is the only one with analog inputs, the data for which triggers event-driven PDOs at a high rate), then the probability of its telegrams being damaged increases. Its error counter will, correspondingly, be the first to reach a critical level.

Node ID / Setting the Baud Rate

Care must be taken to ensure that node addresses are not assigned twice: there may only be one sender for each CAN data telegram.

Test 1

Check node addresses. If the CAN communication functions at least some of the time, and if all the devices support the boot up message, then the address assignment can also be examined by recording the boot up messages after the devices are switched on. This will not, however, recognize node addresses that have been swapped.

Test 2

Check that the same baud rate has been set everywhere. For special devices, if the bit timing parameters are accessible, do they agree with the CANopen definitions (sampling time, SJW, oscillator).

Testing the CAN wiring

These tests should not be carried out if the network is active: No communication should take place during the tests. The following tests should be carried out in the stated sequence, because some of the tests assume that the previous test was successful. Not all the tests are generally necessary.

Network terminator and signal leads

The nodes should be switched off or the CAN cable unplugged for this test, because the results of the measurements can otherwise be distorted by the active CAN transceiver.

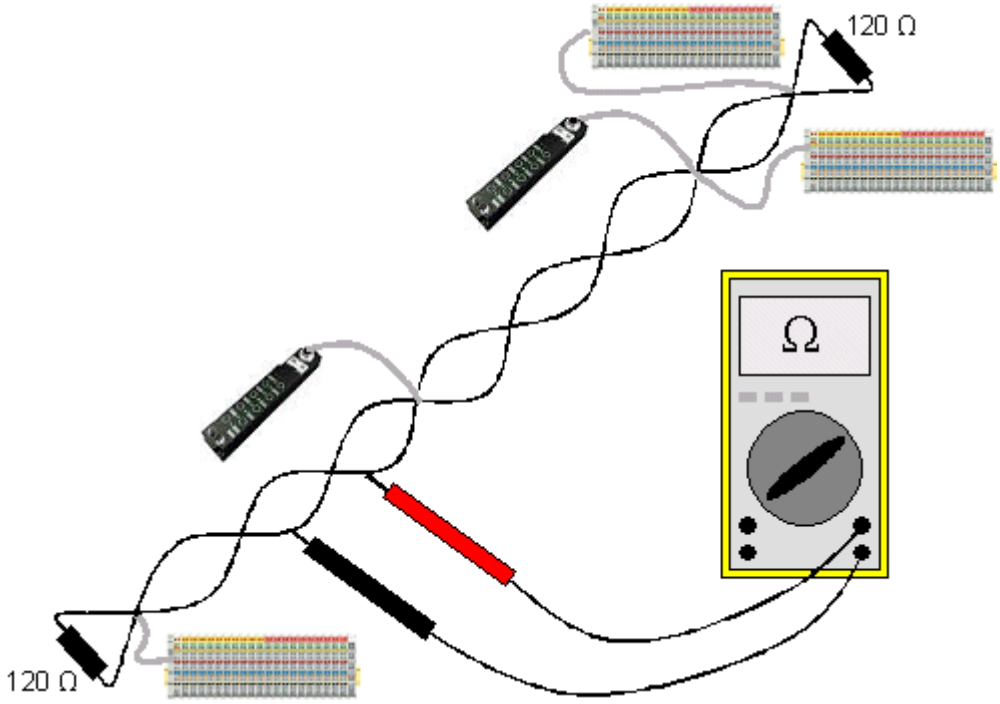


Fig. 139: Wiring diagram for test setup

Test 3

Determine the resistance between CAN high and CAN low - at each device, if necessary.

If the measured value is greater than 65 Ohms, it indicates the absence of a terminating resistor or a break in a signal lead. If the measured value is less than 50 Ohms, look for a short circuit between the CAN lines, more than the correct number of terminating resistors, or faulty transceivers.

Test 4

Check for a short circuit between the CAN ground and the signal leads, or between the screen and signal leads.

Test 5

Remove the earth connection from the CAN ground and screen. Check for a short circuit between the CAN ground and screen.

Topology

The possible cable length in CAN networks depends heavily on the selected baud rate. CAN will tolerate short drop lines - although this again depends on the baud rate. The maximum permitted drop line length should not be exceeded. The length of cable that has been installed is often underestimated - estimates can even be a factor of 10 less than the actual length. The following test is therefore recommended:

Test 6

Measure the lengths of the drop lines and the total bus lengths (do not just make rough estimates!) and compare them with the topology rules for the relevant baud rate.

Screening and earthing

The power supply and the screen should be carefully earthed at the power supply unit, once only and with low resistance. At all connecting points, branches and so forth the screen of the CAN cable (and possibly the CAN GND) must also be connected, as well as the signal leads. In the Beckhoff IP20 Bus Couplers, the screen is grounded for high frequencies via an R/C element.

Test 7

Use a DC ammeter (16 amp max.) to measure the current between the power supply ground and the shield at the end of the network most remote from the power supply unit. An equalization current should be present. If there is no current, then either the screen is not connected all the way through, or the power supply unit is not properly earthed. If the power supply unit is somewhere in the middle of the network, the measurement should be performed at both ends. When appropriate, this test can also be carried out at the ends of the drop line.

Test 8

Interrupt the screen at a number of locations and measure the connection current. If current is flowing, the screen is earthed at more than one place, creating a ground loop.

Potential differences

The screen must be connected all the way through for this test, and must not be carrying any current - this has previously been tested.

Test 9

Measure and record the voltage between the screen and the power supply ground at each node. The maximum potential difference between any two devices should be less than 5 volts.

Detect and localize faults

The "low-tech approach" usually works best: disconnect parts of the network, and observe when the fault disappears.

However, this does not work well for problems such as excessive potential differences, ground loops, EMC or signal distortion, since the reduction in the size of the network often solves the problem without the "missing" piece being the cause. The bus load also changes as the network is reduced in size, which can mean that external interference "hits" CAN telegrams less often.

Diagnosis with an oscilloscope is not usually successful: even when they are in good condition, CAN signals can look really chaotic. It may be possible to trigger on error frames using a storage oscilloscope - this type of diagnosis, however, is only possible for expert technicians.

Protocol problems

In rare cases, protocol problems (e.g. faulty or incomplete CANopen implementation, unfavorable timing at boot up, etc.) can be the cause of faults. In this case it is necessary to trace the bus traffic for evaluation by a CANopen experts - the Beckhoff support team can help here.
A free channel on a Beckhoff FC5102 CANopen PCI card is appropriate for such a trace - Beckhoff make the necessary trace software available on the internet. Alternatively, it is of course possible to use a normal commercial CAN analysis tool.

Protocol problems can be avoided if devices that have not been conformance tested are not used. The official CANopen Conformance Test (and the appropriate certificate) can be obtained from the CAN in Automation Association (http://www.can-cia.de).

7 Appendix

7.1 EtherCAT AL Status Codes

For detailed information please refer to the EtherCAT system description.

7.2 Firmware compatibility

Beckhoff EtherCAT devices are delivered with the latest available firmware version. Compatibility of firmware and hardware is mandatory; not every combination ensures compatibility. The overview below shows the hardware versions on which a firmware can be operated.

Note

- It is recommended to use the newest possible firmware for the respective hardware
- Beckhoff is not under any obligation to provide customers with free firmware updates for delivered products.

NOTE

Risk of damage to the device!

Pay attention to the instructions for firmware updates on the separate page [▶ 177]. If a device is placed in BOOTSTRAP mode for a firmware update, it does not check when downloading whether the new firmware is suitable. This can result in damage to the device! Therefore, always make sure that the firmware is suitable for the hardware version!

EL6751-0000			
Hardware (HW)	**Firmware**	**Revision no.**	**Release date**
07 - 19	06	EL6751-0000-0016	2007/10
	07		2008/11
	08		2008/12
	09	EL6751-0000-0017	2010/06
	10		2010/08
	11		2011/01
		EL6751-0000-0018	2011/02
	12		2012/02
		EL6751-0000-0019	2012/10
	13		2013/03
	14	EL6751-0000-0020	2014/07
20 - 22*	15	EL6751-0000-0021	2014/12
	16	EL6751-0000-0022	2016/04
	17		2017/03
	18*		2018/04

EL6751-0010			
Hardware (HW)	**Firmware**	**Revision no.**	**Release date**
06 - 07	01	EL6751-0010-0016	2007/10
	02	EL6751-0010-0018	2008/11
08 - 18	03		2009/07
	04		2012/03
		EL6751-0010-0019	2012/10
	05	EL6751-0010-0020	2014/07
19 – 21*	06	EL6751-0010-0021	2014/12
	07		2016/04
	08		2017/03
	09*		2018/04

*) This is the current compatible firmware/hardware version at the time of the preparing this documentation. Check on the Beckhoff web page whether more up-to-date documentation is available.

7.3 Firmware Update EL/ES/EM/ELM/EPxxxx

This section describes the device update for Beckhoff EtherCAT slaves from the EL/ES, ELM, EM, EK and EP series. A firmware update should only be carried out after consultation with Beckhoff support.

Storage locations

An EtherCAT slave stores operating data in up to 3 locations:

- Depending on functionality and performance EtherCAT slaves have one or several local controllers for processing I/O data. The corresponding program is the so-called **firmware** in *.efw format.
- In some EtherCAT slaves the EtherCAT communication may also be integrated in these controllers. In this case the controller is usually a so-called **FPGA** chip with *.rbf firmware.
- In addition, each EtherCAT slave has a memory chip, a so-called **ESI-EEPROM**, for storing its own device description (ESI: EtherCAT Slave Information). On power-up this description is loaded and the EtherCAT communication is set up accordingly. The device description is available from the download area of the Beckhoff website at (https://www.beckhoff.de). All ESI files are accessible there as zip files.

Customers can access the data via the EtherCAT fieldbus and its communication mechanisms. Acyclic mailbox communication or register access to the ESC is used for updating or reading of these data.

The TwinCAT System Manager offers mechanisms for programming all 3 parts with new data, if the slave is set up for this purpose. Generally the slave does not check whether the new data are suitable, i.e. it may no longer be able to operate if the data are unsuitable.

Simplified update by bundle firmware

The update using so-called **bundle firmware** is more convenient: in this case the controller firmware and the ESI description are combined in a *.efw file; during the update both the firmware and the ESI are changed in the terminal. For this to happen it is necessary

- for the firmware to be in a packed format: recognizable by the file name, which also contains the revision number, e.g. ELxxxx-xxxx_REV0016_SW01.efw
- for password=1 to be entered in the download dialog. If password=0 (default setting) only the firmware update is carried out, without an ESI update.
- for the device to support this function. The function usually cannot be retrofitted; it is a component of many new developments from year of manufacture 2016.

Following the update, its success should be verified

- ESI/Revision: e.g. by means of an online scan in TwinCAT ConfigMode/FreeRun – this is a convenient way to determine the revision

- Firmware: e.g. by looking in the online CoE of the device

NOTE

Risk of damage to the device!

Note the following when downloading new device files

• Firmware downloads to an EtherCAT device must not be interrupted
• Flawless EtherCAT communication must be ensured. CRC errors or LostFrames must be avoided.
• The power supply must adequately dimensioned. The signal level must meet the specification.

In the event of malfunctions during the update process the EtherCAT device may become unusable and require re-commissioning by the manufacturer.

7.3.1 Device description ESI file/XML

NOTE

Attention regarding update of the ESI description/EEPROM

Some slaves have stored calibration and configuration data from the production in the EEPROM. These are irretrievably overwritten during an update.

The ESI device description is stored locally on the slave and loaded on start-up. Each device description has a unique identifier consisting of slave name (9 characters/digits) and a revision number (4 digits). Each slave configured in the System Manager shows its identifier in the EtherCAT tab:

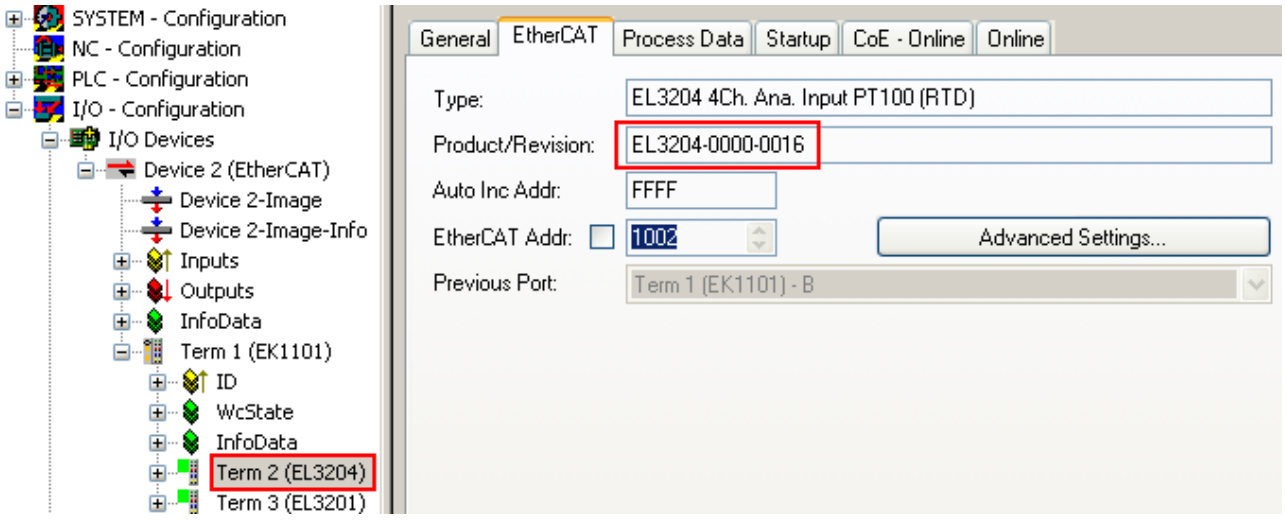


Fig. 140: Device identifier consisting of name EL3204-0000 and revision -0016

The configured identifier must be compatible with the actual device description used as hardware, i.e. the description which the slave has loaded on start-up (in this case EL3204). Normally the configured revision must be the same or lower than that actually present in the terminal network.

For further information on this, please refer to the EtherCAT system documentation.

Update of XML/ESI description

The device revision is closely linked to the firmware and hardware used. Incompatible combinations lead to malfunctions or even final shutdown of the device. Corresponding updates should only be carried out in consultation with Beckhoff support.

Display of ESI slave identifier

The simplest way to ascertain compliance of configured and actual device description is to scan the EtherCAT boxes in TwinCAT mode Config/FreeRun:

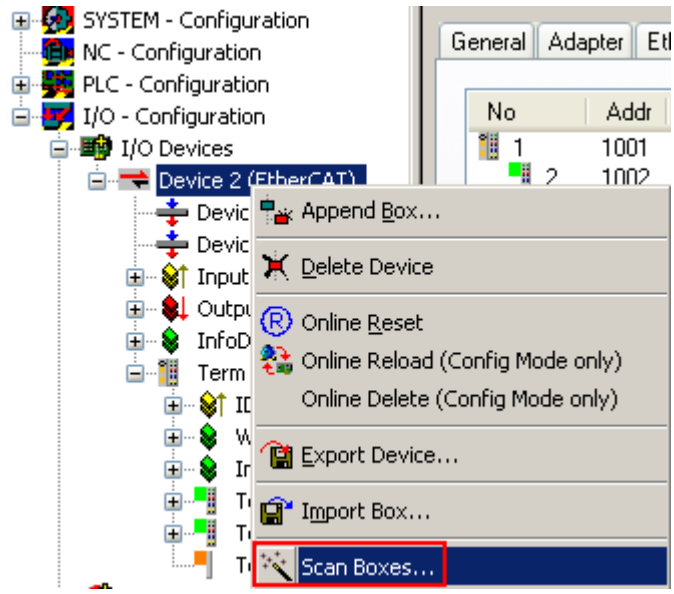


Fig. 141: Scan the subordinate field by right-clicking on the EtherCAT device

If the found field matches the configured field, the display shows

Fig. 142: Configuration is identical

otherwise a change dialog appears for entering the actual data in the configuration.

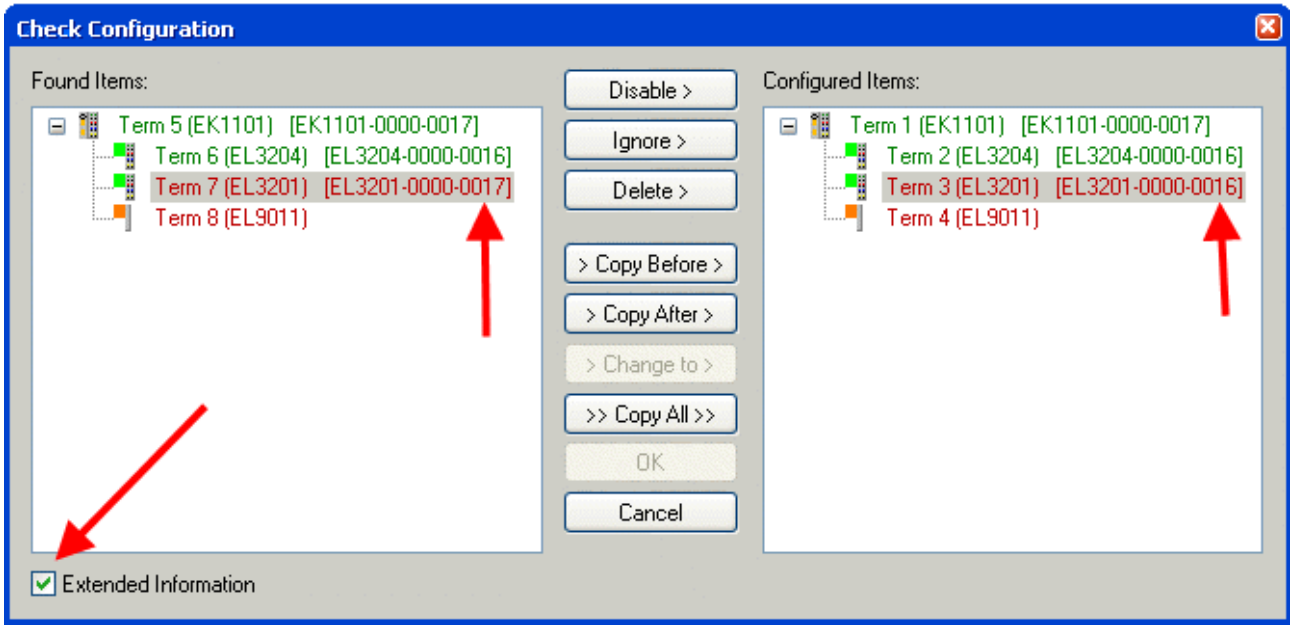


Fig. 143: Change dialog

In this example in Fig. *Change dialog*, an EL3201-0000-**0017** was found, while an EL3201-0000-**0016** was configured. In this case the configuration can be adapted with the *Copy Before* button. The *Extended Information* checkbox must be set in order to display the revision.

Changing the ESI slave identifier

The ESI/EEPROM identifier can be updated as follows under TwinCAT:

- Trouble-free EtherCAT communication must be established with the slave.
- The state of the slave is irrelevant.
- Right-clicking on the slave in the online display opens the *EEPROM Update* dialog, Fig. *EEPROM Update*

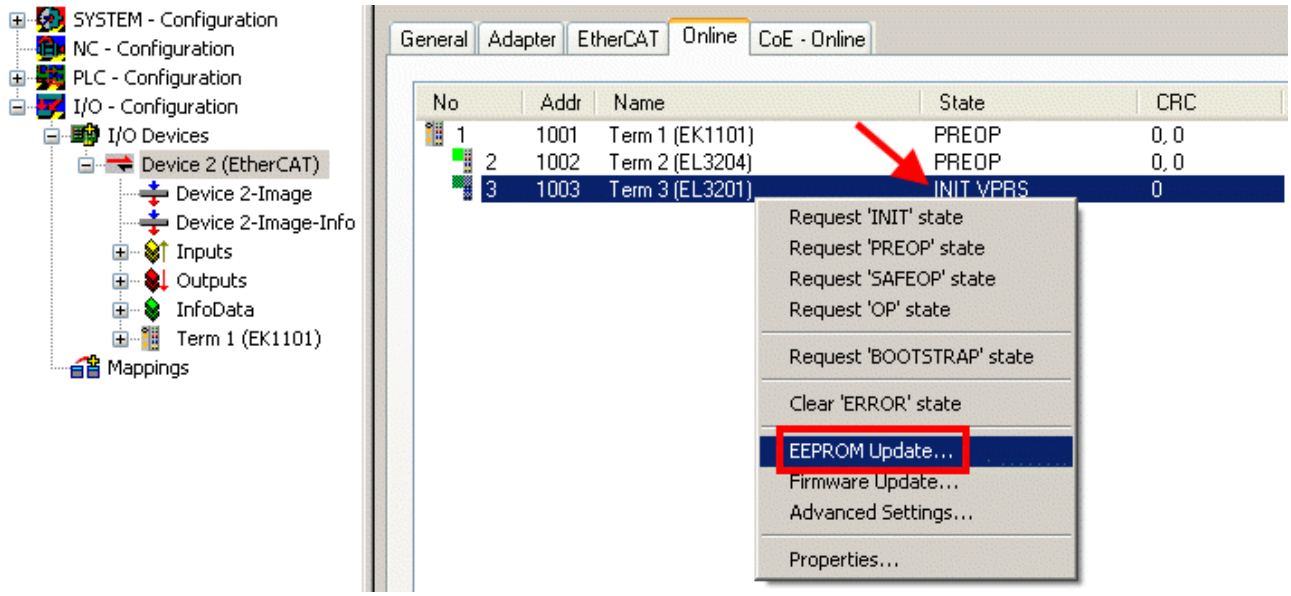


Fig. 144: EEPROM Update

The new ESI description is selected in the following dialog, see Fig. *Selecting the new ESI.* The checkbox *Show Hidden Devices* also displays older, normally hidden versions of a slave.

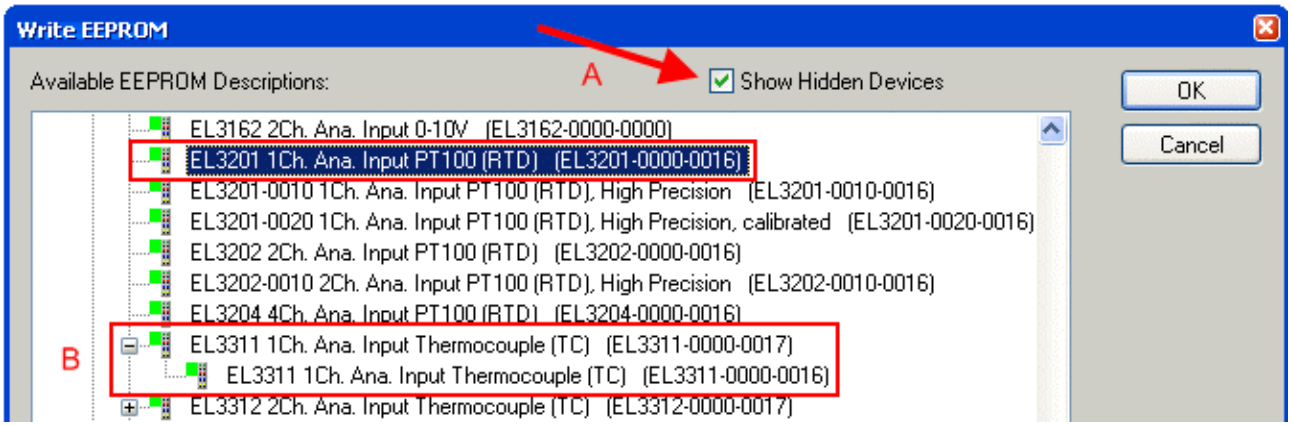


Fig. 145: Selecting the new ESI

A progress bar in the System Manager shows the progress. Data are first written, then verified.

The change only takes effect after a restart.

Most EtherCAT devices read a modified ESI description immediately or after startup from the INIT. Some communication settings such as distributed clocks are only read during power-on. The EtherCAT slave therefore has to be switched off briefly in order for the change to take effect.

7.3.2 Firmware explanation

Determining the firmware version

Determining the version on laser inscription

Beckhoff EtherCAT slaves feature serial numbers applied by laser. The serial number has the following structure: **KK YY FF HH**

KK - week of production (CW, calendar week)
YY - year of production
FF - firmware version
HH - hardware version

Example with ser. no.: 12 10 03 02:

12 - week of production 12
10 - year of production 2010
03 - firmware version 03
02 - hardware version 02

Determining the version via the System Manager

The TwinCAT System Manager shows the version of the controller firmware if the master can access the slave online. Click on the E-Bus Terminal whose controller firmware you want to check (in the example terminal 2 (EL3204)) and select the tab *CoE Online* (CAN over EtherCAT).

CoE Online and Offline CoE

Two CoE directories are available:
• **online**: This is offered in the EtherCAT slave by the controller, if the EtherCAT slave supports this. This CoE directory can only be displayed if a slave is connected and operational.
• **offline**: The EtherCAT Slave Information ESI/XML may contain the default content of the CoE. This CoE directory can only be displayed if it is included in the ESI (e.g. "Beckhoff EL5xxx.xml").

The Advanced button must be used for switching between the two views.

In Fig. *Display of EL3204 firmware version* the firmware version of the selected EL3204 is shown as 03 in CoE entry 0x100A.

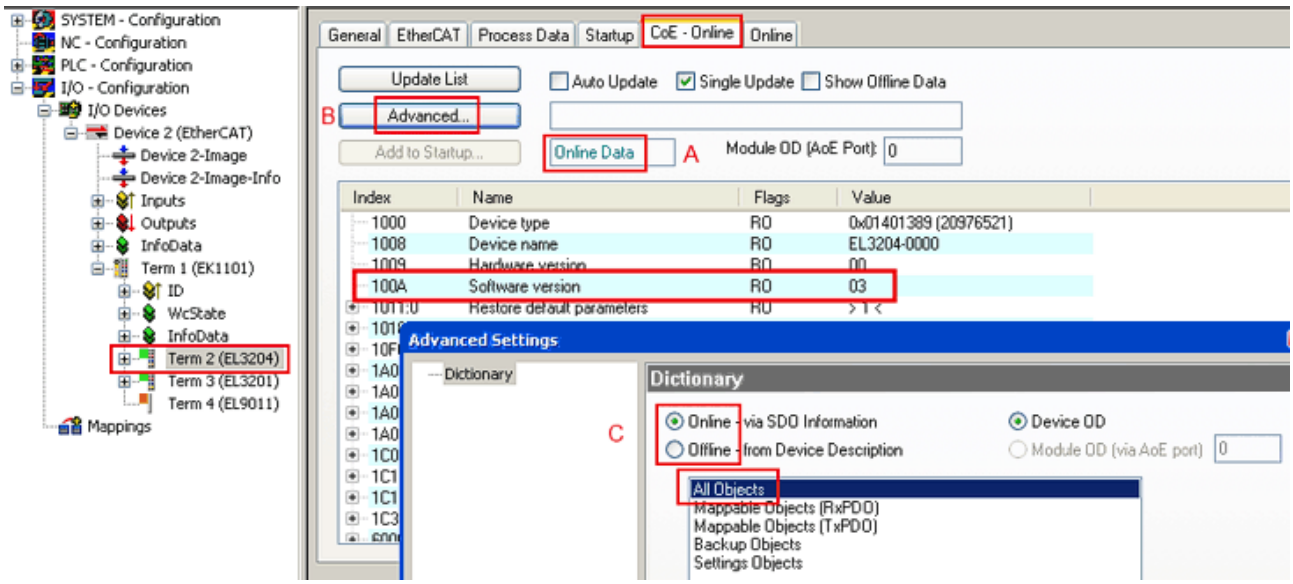


Fig. 146: Display of EL3204 firmware version

In (A) TwinCAT 2.11 shows that the Online CoE directory is currently displayed. If this is not the case, the Online directory can be loaded via the *Online* option in Advanced Settings (B) and double-clicking on *AllObjects*.

7.3.3 Updating controller firmware *.efw

CoE directory

The Online CoE directory is managed by the controller and stored in a dedicated EEPROM, which is generally not changed during a firmware update.

Switch to the *Online* tab to update the controller firmware of a slave, see Fig. *Firmware Update.*

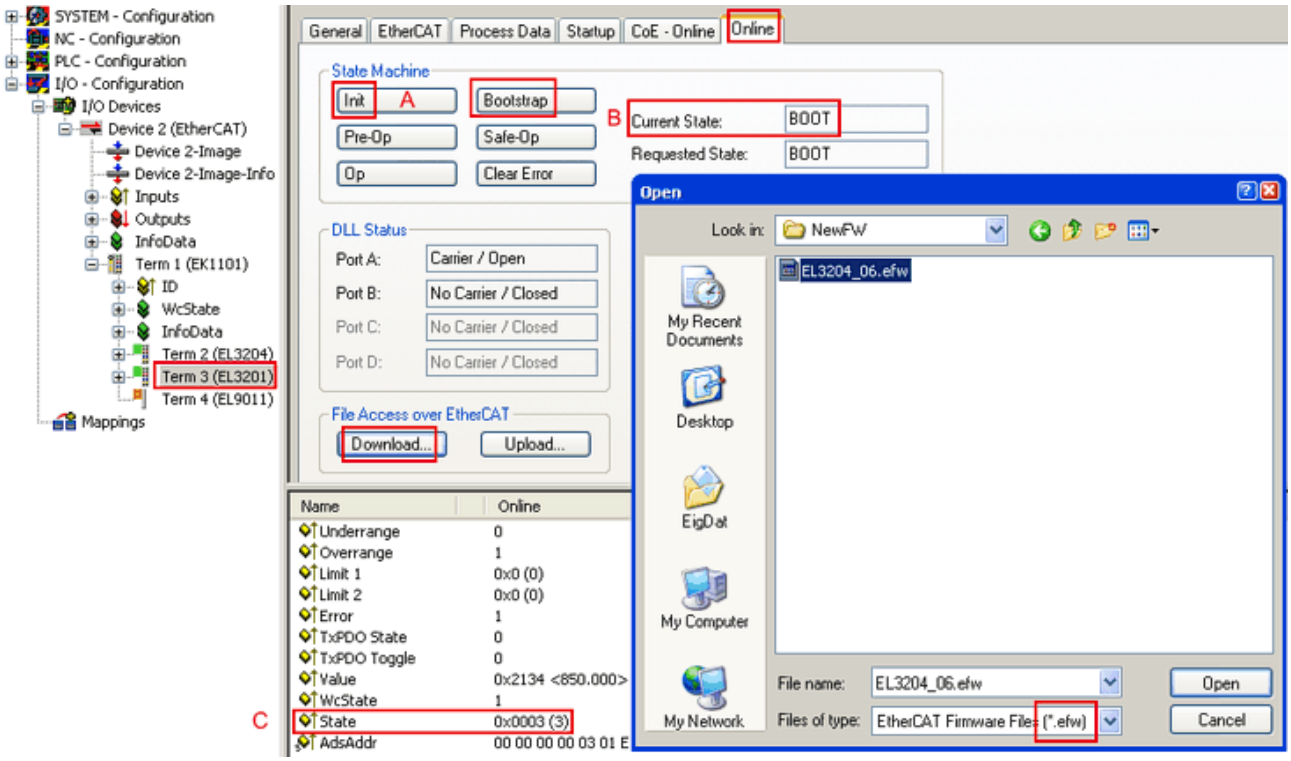
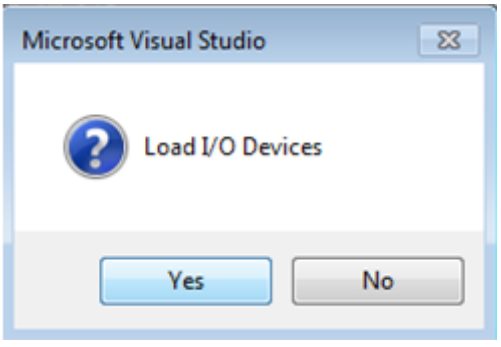
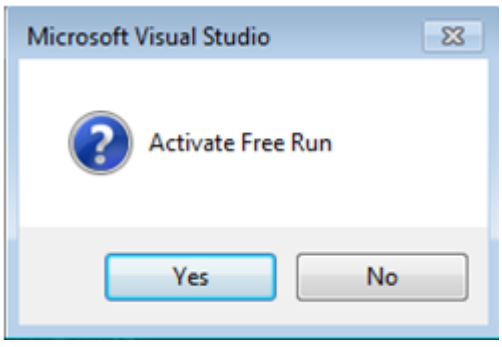


Fig. 147: Firmware Update

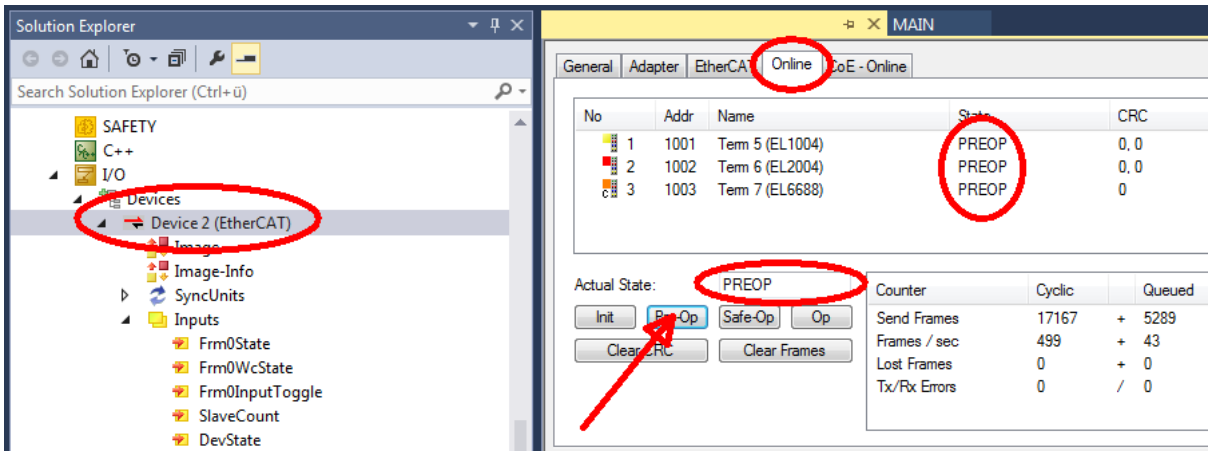
Proceed as follows, unless instructed otherwise by Beckhoff support. Valid for TwinCAT 2 and 3 as EtherCAT master.

- Switch TwinCAT system to ConfigMode/FreeRun with cycle time >= 1 ms (default in ConfigMode is 4 ms). A FW-Update during real time operation is not recommended.

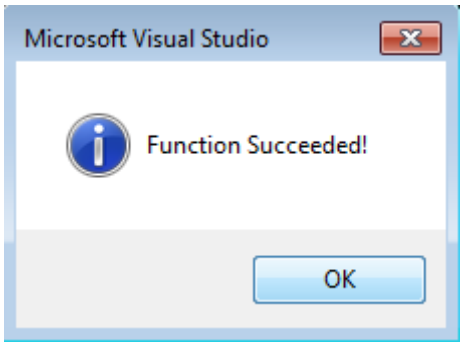




- Switch EtherCAT Master to PreOP



- Switch slave to INIT (A)
- Switch slave to BOOTSTRAP
- Check the current status (B, C)
- Download the new *efw file (wait until it ends). A pass word will not be neccessary usually.



- After the download switch to INIT, then PreOP
- Switch off the slave briefly (don't pull under voltage!)
- Check within CoE 0x100A, if the FW status was correctly overtaken.

7.3.4 FPGA firmware *.rbf

If an FPGA chip deals with the EtherCAT communication an update may be accomplished via an *.rbf file.

- Controller firmware for processing I/O signals
- FPGA firmware for EtherCAT communication (only for terminals with FPGA)

The firmware version number included in the terminal serial number contains both firmware components. If one of these firmware components is modified this version number is updated.

Determining the version via the System Manager

The TwinCAT System Manager indicates the FPGA firmware version. Click on the Ethernet card of your EtherCAT strand (Device 2 in the example) and select the *Online* tab.

The *Reg:0002* column indicates the firmware version of the individual EtherCAT devices in hexadecimal and decimal representation.

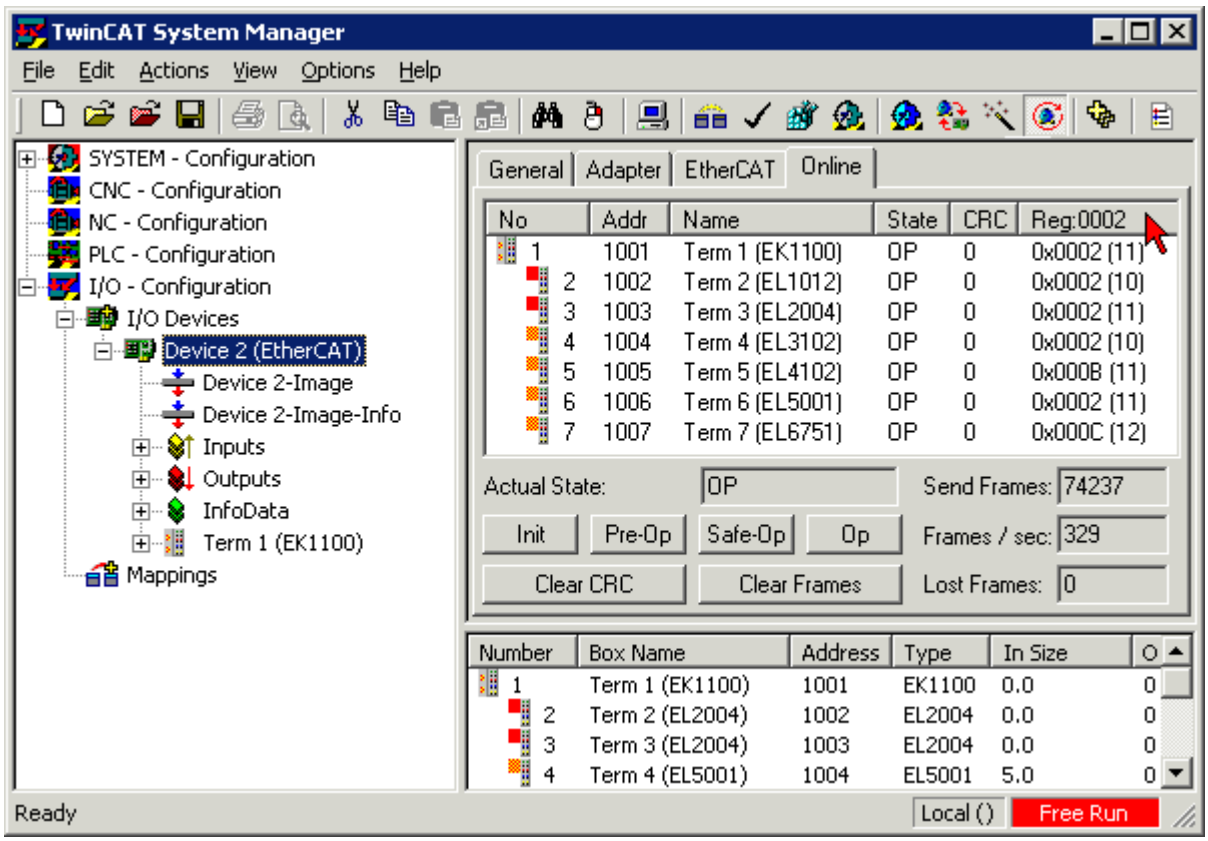


Fig. 148: FPGA firmware version definition

If the column *Reg:0002* is not displayed, right-click the table header and select *Properties* in the context menu.

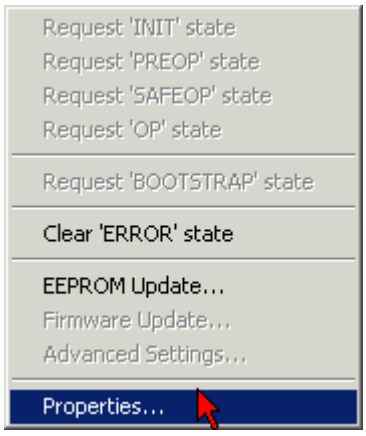


Fig. 149: Context menu *Properties*

The *Advanced Settings* dialog appears where the columns to be displayed can be selected. Under *Diagnosis/***Online View** select the *'0002 ETxxxx Build'* check box in order to activate the FPGA firmware version display.

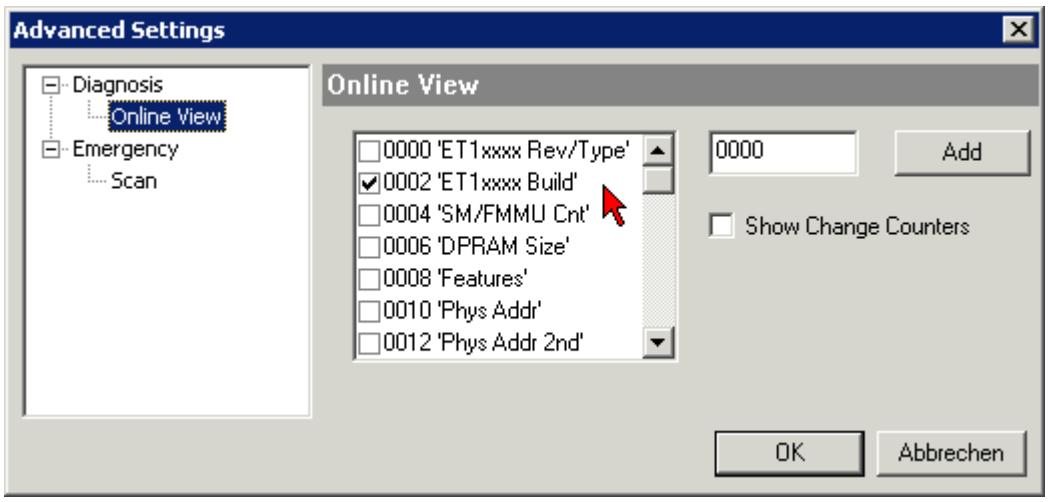


Fig. 150: Dialog *Advanced Settings*

Update

For updating the FPGA firmware

- of an EtherCAT coupler the coupler must have FPGA firmware version 11 or higher;
- of an E-Bus Terminal the terminal must have FPGA firmware version 10 or higher.

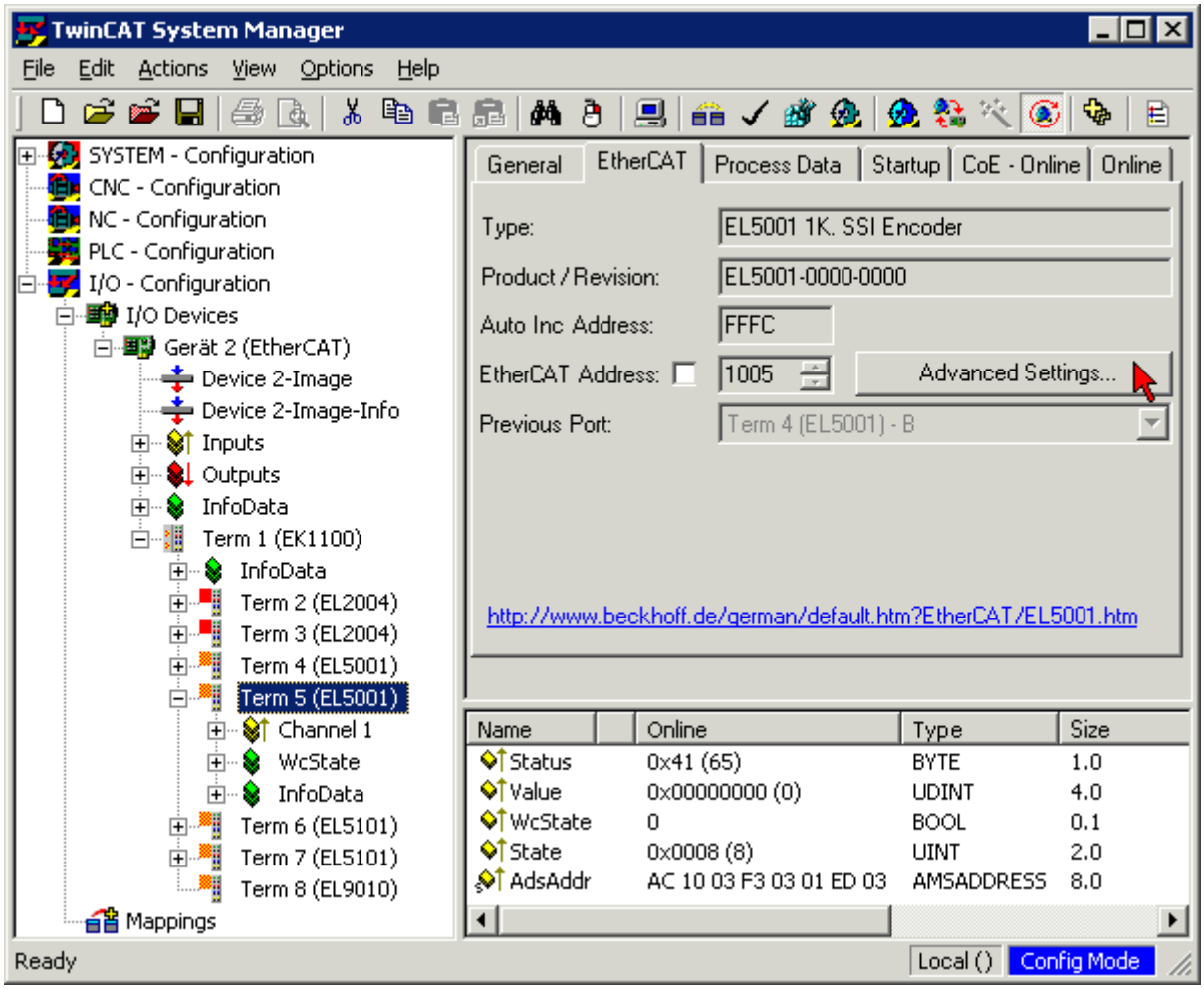
Older firmware versions can only be updated by the manufacturer!

Updating an EtherCAT device

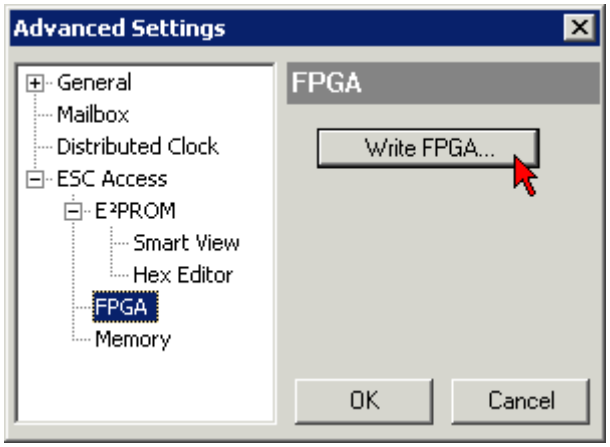
The following sequence order have to be met if no other specifications are given (e.g. by the Beckhoff support):

- Switch TwinCAT system to ConfigMode/FreeRun with cycle time >= 1 ms (default in ConfigMode is 4 ms). A FW-Update during real time operation is not recommended.

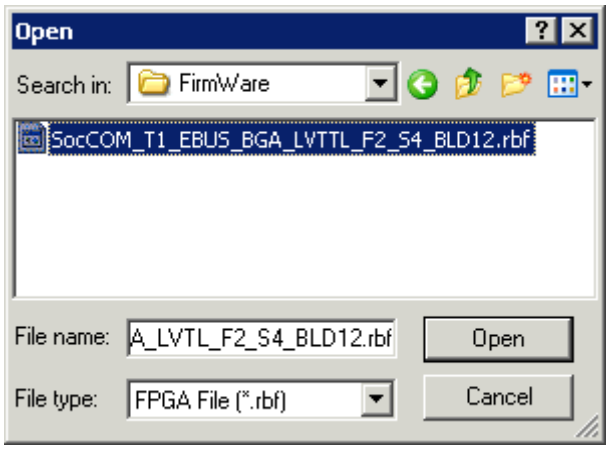
- In the TwinCAT System Manager select the terminal for which the FPGA firmware is to be updated (in the example: Terminal 5: EL5001) and
  click the *Advanced Settings* button in the *EtherCAT* tab:



- The *Advanced Settings* dialog appears. Under *ESC Access/E²PROM*/FPGA click on *Write FPGA* button:



- Select the file (*.rbf) with the new FPGA firmware, and transfer it to the EtherCAT device:



- Wait until download ends
- Switch slave current less for a short time (don't pull under voltage!). In order to activate the new FPGA firmware a restart (switching the power supply off and on again) of the EtherCAT device is required.
- Check the new FPGA status

NOTE

Risk of damage to the device!

A download of firmware to an EtherCAT device must not be interrupted in any case! If you interrupt this process by switching off power supply or disconnecting the Ethernet link, the EtherCAT device can only be recommissioned by the manufacturer!

7.3.5 Simultaneous updating of several EtherCAT devices

The firmware and ESI descriptions of several devices can be updated simultaneously, provided the devices have the same firmware file/ESI.

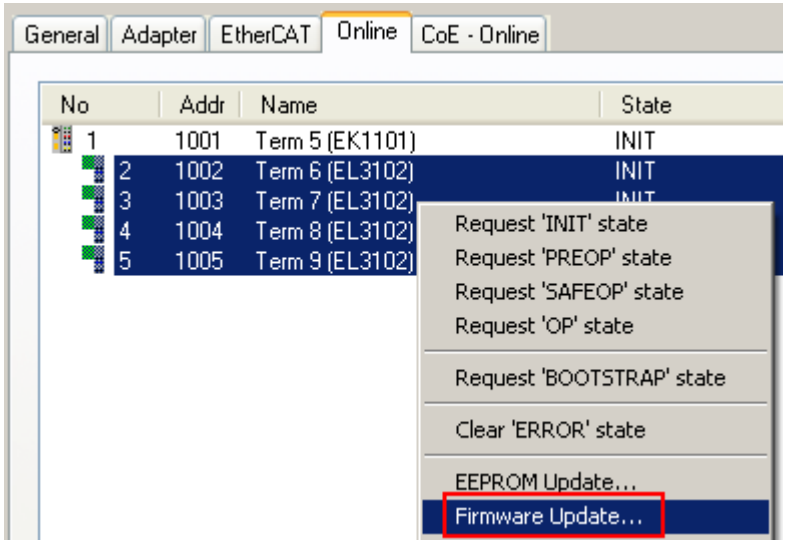


Fig. 151: Multiple selection and firmware update

Select the required slaves and carry out the firmware update in BOOTSTRAP mode as described above.

7.4 CAN Identifier List

The list provided here should assist in identifying and assigning CANopen messages. All the identifiers allocated by the CANopen default identifier allocation are listed, as well as the manufacturer-specific default identifiers issued by BECKHOFF via object 0x5500 (only to be used in networks with node addresses less than 64).

The following values can be used as search aids and "entry points" in the extensive identifier table in the *chm edition of the documentation:

Decimal: 400 [▶ 189], 500 [▶ 196], 600 [▶ 196], 700 [▶ 191], 800 [▶ 192], 900 [▶ 193], 1000 [▶ 198], 1100 [▶ 198], 1200 [▶ 194], 1300 [▶ 194], 1400 [▶ 199], 1500 [▶ 200], 1600 [▶ 200], 1700 [▶ 195], 1800 [▶ 203], 1900 [▶ 195]

Hexadecimal: 0x181 [▶ 189], 0x1C1 [▶ 196], 0x201 [▶ 190], 0x301 [▶ 192], 0x401 [▶ 193], 0x501 [▶ 194], 0x601 [▶ 202], 0x701 [▶ 203]

The identifier distribution via object 0x5500 follows this pattern:

Object	Resulting COB ID (dec)	Resulting COB ID (hex)
Emergency [▶ 189]	129 to 191 [255]	0x81 to 0xBF [0xFF]
TxPDO1 [▶ 189]	385 to 447 [511]	0x181 to 0x1BF [0x1FF]
RxPDO1 [▶ 190]	513 to 575 [639]	0x201 to 0x23F [0x27F]
TxPDO2 [▶ 191]	641 to 676 [767]	0x281 to 0x2BF [0x2FF]
RxPDO2 [▶ 192]	769 to 831 [895]	0x301 to 0x33F [0x37F]
TxDPO3 [▶ 193]	897 to 959 [1023]	0x381 to 0x3BF [0x3FF]
RxPDO3 [▶ 193]	1025 to 1087 [1151]	0x401 to 0x43F [0x47F]
TxPDO4 [▶ 194]	1153 to 1215 [1279]	0x481 to 0x4BF [0x4FF]
RxPDO4 [▶ 194]	1281 to 1343 [1407]	0x501 to 0x53F [0x57F]
TxPDO5 [▶ 195]	1665 to 1727	0x681 to 0x6BF
RxPDO5 [▶ 195]	1921 to 1983	0x781 to 0x7BF
TxPDO6 [▶ 196]	449 to 511	0x1C1 to 0x1FF
RxPDO6 [▶ 196]	577 to 639	0x241 to 0x27F
TxDPO7 [▶ 197]	705 to 767	0x2C1 to 0x2FF
RxPDO7 [▶ 197]	833 to 895	0x341 to 0x37F
TxPDO8 [▶ 198]	961 to 1023	0x3C1 to 0x3FF
RxPDO8 [▶ 198]	1089 to 1151	0x441 to 0x47F
TxPDO9 [▶ 199]	1217 to 1279	0x4C1 to 0x4FF
RxPDO9 [▶ 199]	1345 to 1407	0x541 to 0x57F
TxDPO10 [▶ 200]	1473 to 1535	0x5C1 to 0x5FF
RxPDO10 [▶ 200]	1601 to 1663	0x641 to 0x67F
TxPDO11 [▶ 201]	1729 to 1791	0x6C1 to 0x6FF
RxPDO11 [▶ 201]	1857 to 1919	0x741 to 0x77F
SDO (Tx) [▶ 202]	1409 to 1471 [1535]	0x581 to 0x5BF [0x5FF]
SDO (Rx) [▶ 202]	1537 to 1599 [1663]	0x601 to 0x63F [0x67F]
Guarding / Heartbeat/ Bootup [▶ 203]	1793 to 1855 [1919]	0x701 to 0x73F [0x77F]

Identifier List

Identifiers marked with * are given manufacturer-specific assignments on the Bus Couplers after writing index 0x5500

dec	hex	Telegram type	dec	hex	Telegram type	dec	hex	Telegram type
0	0x00	NMT	149	0x95	EMCY Nd.21	171	0xAB	EMCY Nd.43
128	0x80	SYNC	150	0x96	EMCY Nd.22	172	0xAC	EMCY Nd.44
129	0x81	EMCY Nd.1	151	0x97	EMCY Nd.23	173	0xAD	EMCY Nd.45
130	0x82	EMCY Nd.2	152	0x98	EMCY Nd.24	174	0xAE	EMCY Nd.46
131	0x83	EMCY Nd.3	153	0x99	EMCY Nd.25	175	0xAF	EMCY Nd.47
132	0x84	EMCY Nd.4	154	0x9A	EMCY Nd.26	176	0xB0	EMCY Nd.48
133	0x85	EMCY Nd.5	155	0x9B	EMCY Nd.27	177	0xB1	EMCY Nd.49
134	0x86	EMCY Nd.6	156	0x9C	EMCY Nd.28	178	0xB2	EMCY Nd.50
135	0x87	EMCY Nd.7	157	0x9D	EMCY Nd.29	179	0xB3	EMCY Nd.51
136	0x88	EMCY Nd.8	158	0x9E	EMCY Nd.30	180	0xB4	EMCY Nd.52
137	0x89	EMCY Nd.9	159	0x9F	EMCY Nd.31	181	0xB5	EMCY Nd.53
138	0x8A	EMCY Nd.10	160	0xA0	EMCY Nd.32	182	0xB6	EMCY Nd.54
139	0x8B	EMCY Nd.11	161	0xA1	EMCY Nd.33	183	0xB7	EMCY Nd.55
140	0x8C	EMCY Nd.12	162	0xA2	EMCY Nd.34	184	0xB8	EMCY Nd.56
141	0x8D	EMCY Nd.13	163	0xA3	EMCY Nd.35	185	0xB9	EMCY Nd.57
142	0x8E	EMCY Nd.14	164	0xA4	EMCY Nd.36	186	0xBA	EMCY Nd.58
143	0x8F	EMCY Nd.15	165	0xA5	EMCY Nd.37	187	0xBB	EMCY Nd.59
144	0x90	EMCY Nd.16	166	0xA6	EMCY Nd.38	188	0xBC	EMCY Nd.60
145	0x91	EMCY Nd.17	167	0xA7	EMCY Nd.39	189	0xBD	EMCY Nd.61
146	0x92	EMCY Nd.18	168	0xA8	EMCY Nd.40	190	0xBE	EMCY Nd.62
147	0x93	EMCY Nd.19	169	0xA9	EMCY Nd.41	191	0xBF	EMCY Nd.63
148	0x94	EMCY Nd.20	170	0xAA	EMCY Nd.42			

dec	hex	Telegram type	dec	hex	Telegram type	dec	hex	Telegram type
385	0x181	TxPDO1, DI, Nd.1	406	0x196	TxPDO1, DI, Nd.22	427	0x1AB	TxPDO1, DI, Nd.43
386	0x182	TxPDO1, DI, Nd.2	407	0x197	TxPDO1, DI, Nd.23	428	0x1AC	TxPDO1, DI, Nd.44
387	0x183	TxPDO1, DI, Nd.3	408	0x198	TxPDO1, DI, Nd.24	429	0x1AD	TxPDO1, DI, Nd.45
388	0x184	TxPDO1, DI, Nd.4	409	0x199	TxPDO1, DI, Nd.25	430	0x1AE	TxPDO1, DI, Nd.46
389	0x185	TxPDO1, DI, Nd.5	410	0x19A	TxPDO1, DI, Nd.26	431	0x1AF	TxPDO1, DI, Nd.47
390	0x186	TxPDO1, DI, Nd.6	411	0x19B	TxPDO1, DI, Nd.27	432	0x1B0	TxPDO1, DI, Nd.48
391	0x187	TxPDO1, DI, Nd.7	412	0x19C	TxPDO1, DI, Nd.28	433	0x1B1	TxPDO1, DI, Nd.49
392	0x188	TxPDO1, DI, Nd.8	413	0x19D	TxPDO1, DI, Nd.29	434	0x1B2	TxPDO1, DI, Nd.50
393	0x189	TxPDO1, DI, Nd.9	414	0x19E	TxPDO1, DI, Nd.30	435	0x1B3	TxPDO1, DI, Nd.51
394	0x18A	TxPDO1, DI, Nd.10	415	0x19F	TxPDO1, DI, Nd.31	436	0x1B4	TxPDO1, DI, Nd.52
395	0x18B	TxPDO1, DI, Nd.11	416	0x1A0	TxPDO1, DI, Nd.32	437	0x1B5	TxPDO1, DI, Nd.53
396	0x18C	TxPDO1, DI, Nd.12	417	0x1A1	TxPDO1, DI, Nd.33	438	0x1B6	TxPDO1, DI, Nd.54
397	0x18D	TxPDO1, DI, Nd.13	418	0x1A2	TxPDO1, DI, Nd.34	439	0x1B7	TxPDO1, DI, Nd.55
398	0x18E	TxPDO1, DI, Nd.14	419	0x1A3	TxPDO1, DI, Nd.35	440	0x1B8	TxPDO1, DI, Nd.56
399	0x18F	TxPDO1, DI, Nd.15	420	0x1A4	TxPDO1, DI, Nd.36	441	0x1B9	TxPDO1, DI, Nd.57
400	0x190	TxPDO1, DI, Nd.16	421	0x1A5	TxPDO1, DI, Nd.37	442	0x1BA	TxPDO1, DI, Nd.58
401	0x191	TxPDO1, DI, Nd.17	422	0x1A6	TxPDO1, DI, Nd.38	443	0x1BB	TxPDO1, DI, Nd.59
402	0x192	TxPDO1, DI, Nd.18	423	0x1A7	TxPDO1, DI, Nd.39	444	0x1BC	TxPDO1, DI, Nd.60
403	0x193	TxPDO1, DI, Nd.19	424	0x1A8	TxPDO1, DI, Nd.40	445	0x1BD	TxPDO1, DI, Nd.61
404	0x194	TxPDO1, DI, Nd.20	425	0x1A9	TxPDO1, DI, Nd.41	446	0x1BE	TxPDO1, DI, Nd.62
405	0x195	TxPDO1, DI, Nd.21	426	0x1AA	TxPDO1, DI, Nd.42	447	0x1BF	TxPDO1, DI, Nd.63

dec	hex	Telegram type	dec	hex	Telegram type	dec	hex	Telegram type
513	0x201	RxPDO1, DO, Nd.1	534	0x216	RxPDO1, DO, Nd.22	555	0x22B	RxPDO1, DO, Nd.43
514	0x202	RxPDO1, DO, Nd.2	535	0x217	RxPDO1, DO, Nd.23	556	0x22C	RxPDO1, DO, Nd.44
515	0x203	RxPDO1, DO, Nd.3	536	0x218	RxPDO1, DO, Nd.24	557	0x22D	RxPDO1, DO, Nd.45
516	0x204	RxPDO1, DO, Nd.4	537	0x219	RxPDO1, DO, Nd.25	558	0x22E	RxPDO1, DO, Nd.46
517	0x205	RxPDO1, DO, Nd.5	538	0x21A	RxPDO1, DO, Nd.26	559	0x22F	RxPDO1, DO, Nd.47
518	0x206	RxPDO1, DO, Nd.6	539	0x21B	RxPDO1, DO, Nd.27	560	0x230	RxPDO1, DO, Nd.48
519	0x207	RxPDO1, DO, Nd.7	540	0x21C	RxPDO1, DO, Nd.28	561	0x231	RxPDO1, DO, Nd.49
520	0x208	RxPDO1, DO, Nd.8	541	0x21D	RxPDO1, DO, Nd.29	562	0x232	RxPDO1, DO, Nd.50
521	0x209	RxPDO1, DO, Nd.9	542	0x21E	RxPDO1, DO, Nd.30	563	0x233	RxPDO1, DO, Nd.51
522	0x20A	RxPDO1, DO, Nd.10	543	0x21F	RxPDO1, DO, Nd.31	564	0x234	RxPDO1, DO, Nd.52
523	0x20B	RxPDO1, DO, Nd.11	544	0x220	RxPDO1, DO, Nd.32	565	0x235	RxPDO1, DO, Nd.53
524	0x20C	RxPDO1, DO, Nd.12	545	0x221	RxPDO1, DO, Nd.33	566	0x236	RxPDO1, DO, Nd.54
525	0x20D	RxPDO1, DO, Nd.13	546	0x222	RxPDO1, DO, Nd.34	567	0x237	RxPDO1, DO, Nd.55
526	0x20E	RxPDO1, DO, Nd.14	547	0x223	RxPDO1, DO, Nd.35	568	0x238	RxPDO1, DO, Nd.56
527	0x20F	RxPDO1, DO, Nd.15	548	0x224	RxPDO1, DO, Nd.36	569	0x239	RxPDO1, DO, Nd.57
528	0x210	RxPDO1, DO, Nd.16	549	0x225	RxPDO1, DO, Nd.37	570	0x23A	RxPDO1, DO, Nd.58
529	0x211	RxPDO1, DO, Nd.17	550	0x226	RxPDO1, DO, Nd.38	571	0x23B	RxPDO1, DO, Nd.59
530	0x212	RxPDO1, DO, Nd.18	551	0x227	RxPDO1, DO, Nd.39	572	0x23C	RxPDO1, DO, Nd.60
531	0x213	RxPDO1, DO, Nd.19	552	0x228	RxPDO1, DO, Nd.40	573	0x23D	RxPDO1, DO, Nd.61
532	0x214	RxPDO1, DO, Nd.20	553	0x229	RxPDO1, DO, Nd.41	574	0x23E	RxPDO1, DO, Nd.62
533	0x215	RxPDO1, DO, Nd.21	554	0x22A	RxPDO1, DO, Nd.42	575	0x23F	RxPDO1, DO, Nd.63

dec	hex	Telegram type	dec	hex	Telegram type	dec	hex	Telegram type
641	0x281	TxPDO2, AI, Nd.1	662	0x296	TxPDO2, AI, Nd.22	683	0x2AB	TxPDO2, AI, Nd.43
642	0x282	TxPDO2, AI, Nd.2	663	0x297	TxPDO2, AI, Nd.23	684	0x2AC	TxPDO2, AI, Nd.44
643	0x283	TxPDO2, AI, Nd.3	664	0x298	TxPDO2, AI, Nd.24	685	0x2AD	TxPDO2, AI, Nd.45
644	0x284	TxPDO2, AI, Nd.4	665	0x299	TxPDO2, AI, Nd.25	686	0x2AE	TxPDO2, AI, Nd.46
645	0x285	TxPDO2, AI, Nd.5	666	0x29A	TxPDO2, AI, Nd.26	687	0x2AF	TxPDO2, AI, Nd.47
646	0x286	TxPDO2, AI, Nd.6	667	0x29B	TxPDO2, AI, Nd.27	688	0x2B0	TxPDO2, AI, Nd.48
647	0x287	TxPDO2, AI, Nd.7	668	0x29C	TxPDO2, AI, Nd.28	689	0x2B1	TxPDO2, AI, Nd.49
648	0x288	TxPDO2, AI, Nd.8	669	0x29D	TxPDO2, AI, Nd.29	690	0x2B2	TxPDO2, AI, Nd.50
649	0x289	TxPDO2, AI, Nd.9	670	0x29E	TxPDO2, AI, Nd.30	691	0x2B3	TxPDO2, AI, Nd.51
650	0x28A	TxPDO2, AI, Nd.10	671	0x29F	TxPDO2, AI, Nd.31	692	0x2B4	TxPDO2, AI, Nd.52
651	0x28B	TxPDO2, AI, Nd.11	672	0x2A0	TxPDO2, AI, Nd.32	693	0x2B5	TxPDO2, AI, Nd.53
652	0x28C	TxPDO2, AI, Nd.12	673	0x2A1	TxPDO2, AI, Nd.33	694	0x2B6	TxPDO2, AI, Nd.54
653	0x28D	TxPDO2, AI, Nd.13	674	0x2A2	TxPDO2, AI, Nd.34	695	0x2B7	TxPDO2, AI, Nd.55
654	0x28E	TxPDO2, AI, Nd.14	675	0x2A3	TxPDO2, AI, Nd.35	696	0x2B8	TxPDO2, AI, Nd.56
655	0x28F	TxPDO2, AI, Nd.15	676	0x2A4	TxPDO2, AI, Nd.36	697	0x2B9	TxPDO2, AI, Nd.57
656	0x290	TxPDO2, AI, Nd.16	677	0x2A5	TxPDO2, AI, Nd.37	698	0x2BA	TxPDO2, AI, Nd.58
657	0x291	TxPDO2, AI, Nd.17	678	0x2A6	TxPDO2, AI, Nd.38	699	0x2BB	TxPDO2, AI, Nd.59
658	0x292	TxPDO2, AI, Nd.18	679	0x2A7	TxPDO2, AI, Nd.39	700	0x2BC	TxPDO2, AI, Nd.60
659	0x293	TxPDO2, AI, Nd.19	680	0x2A8	TxPDO2, AI, Nd.40	701	0x2BD	TxPDO2, AI, Nd.61
660	0x294	TxPDO2, AI, Nd.20	681	0x2A9	TxPDO2, AI, Nd.41	702	0x2BE	TxPDO2, AI, Nd.62
661	0x295	TxPDO2, AI, Nd.21	682	0x2AA	TxPDO2, AI, Nd.42	703	0x2BF	TxPDO2, AI, Nd.63

dec	hex	Telegram type	dec	hex	Telegram type	dec	hex	Telegram type
769	0x301	RxPDO2, AO, Nd.1	790	0x316	RxPDO2, AO, Nd.22	811	0x32B	RxPDO2, AO, Nd.43
770	0x302	RxPDO2, AO, Nd.2	791	0x317	RxPDO2, AO, Nd.23	812	0x32C	RxPDO2, AO, Nd.44
771	0x303	RxPDO2, AO, Nd.3	792	0x318	RxPDO2, AO, Nd.24	813	0x32D	RxPDO2, AO, Nd.45
772	0x304	RxPDO2, AO, Nd.4	793	0x319	RxPDO2, AO, Nd.25	814	0x32E	RxPDO2, AO, Nd.46
773	0x305	RxPDO2, AO, Nd.5	794	0x31A	RxPDO2, AO, Nd.26	815	0x32F	RxPDO2, AO, Nd.47
774	0x306	RxPDO2, AO, Nd.6	795	0x31B	RxPDO2, AO, Nd.27	816	0x330	RxPDO2, AO, Nd.48
775	0x307	RxPDO2, AO, Nd.7	796	0x31C	RxPDO2, AO, Nd.28	817	0x331	RxPDO2, AO, Nd.49
776	0x308	RxPDO2, AO, Nd.8	797	0x31D	RxPDO2, AO, Nd.29	818	0x332	RxPDO2, AO, Nd.50
777	0x309	RxPDO2, AO, Nd.9	798	0x31E	RxPDO2, AO, Nd.30	819	0x333	RxPDO2, AO, Nd.51
778	0x30A	RxPDO2, AO, Nd.10	799	0x31F	RxPDO2, AO, Nd.31	820	0x334	RxPDO2, AO, Nd.52
779	0x30B	RxPDO2, AO, Nd.11	800	0x320	RxPDO2, AO, Nd.32	821	0x335	RxPDO2, AO, Nd.53
780	0x30C	RxPDO2, AO, Nd.12	801	0x321	RxPDO2, AO, Nd.33	822	0x336	RxPDO2, AO, Nd.54
781	0x30D	RxPDO2, AO, Nd.13	802	0x322	RxPDO2, AO, Nd.34	823	0x337	RxPDO2, AO, Nd.55
782	0x30E	RxPDO2, AO, Nd.14	803	0x323	RxPDO2, AO, Nd.35	824	0x338	RxPDO2, AO, Nd.56
783	0x30F	RxPDO2, AO, Nd.15	804	0x324	RxPDO2, AO, Nd.36	825	0x339	RxPDO2, AO, Nd.57
784	0x310	RxPDO2, AO, Nd.16	805	0x325	RxPDO2, AO, Nd.37	826	0x33A	RxPDO2, AO, Nd.58
785	0x311	RxPDO2, AO, Nd.17	806	0x326	RxPDO2, AO, Nd.38	827	0x33B	RxPDO2, AO, Nd.59
786	0x312	RxPDO2, AO, Nd.18	807	0x327	RxPDO2, AO, Nd.39	828	0x33C	RxPDO2, AO, Nd.60
787	0x313	RxPDO2, AO, Nd.19	808	0x328	RxPDO2, AO, Nd.40	829	0x33D	RxPDO2, AO, Nd.61
788	0x314	RxPDO2, AO, Nd.20	809	0x329	RxPDO2, AO, Nd.41	830	0x33E	RxPDO2, AO, Nd.62
789	0x315	RxPDO2, AO, Nd.21	810	0x32A	RxPDO2, AO, Nd.42	831	0x33F	RxPDO2, AO, Nd.63

dec	hex	Telegram type	dec	hex	Telegram type	dec	hex	Telegram type
897	0x381	TxPDO3*, Nd.1	918	0x396	TxPDO3*, Nd.22	939	0x3AB	TxPDO3*, Nd.43
898	0x382	TxPDO3*, Nd.2	919	0x397	TxPDO3*, Nd.23	940	0x3AC	TxPDO3*, Nd.44
899	0x383	TxPDO3*, Nd.3	920	0x398	TxPDO3*, Nd.24	941	0x3AD	TxPDO3*, Nd.45
900	0x384	TxPDO3*, Nd.4	921	0x399	TxPDO3*, Nd.25	942	0x3AE	TxPDO3*, Nd.46
901	0x385	TxPDO3*, Nd.5	922	0x39A	TxPDO3*, Nd.26	943	0x3AF	TxPDO3*, Nd.47
902	0x386	TxPDO3*, Nd.6	923	0x39B	TxPDO3*, Nd.27	944	0x3B0	TxPDO3*, Nd.48
903	0x387	TxPDO3*, Nd.7	924	0x39C	TxPDO3*, Nd.28	945	0x3B1	TxPDO3*, Nd.49
904	0x388	TxPDO3*, Nd.8	925	0x39D	TxPDO3*, Nd.29	946	0x3B2	TxPDO3*, Nd.50
905	0x389	TxPDO3*, Nd.9	926	0x39E	TxPDO3*, Nd.30	947	0x3B3	TxPDO3*, Nd.51
906	0x38A	TxPDO3*, Nd.10	927	0x39F	TxPDO3*, Nd.31	948	0x3B4	TxPDO3*, Nd.52
907	0x38B	TxPDO3*, Nd.11	928	0x3A0	TxPDO3*, Nd.32	949	0x3B5	TxPDO3*, Nd.53
908	0x38C	TxPDO3*, Nd.12	929	0x3A1	TxPDO3*, Nd.33	950	0x3B6	TxPDO3*, Nd.54
909	0x38D	TxPDO3*, Nd.13	930	0x3A2	TxPDO3*, Nd.34	951	0x3B7	TxPDO3*, Nd.55
910	0x38E	TxPDO3*, Nd.14	931	0x3A3	TxPDO3*, Nd.35	952	0x3B8	TxPDO3*, Nd.56
911	0x38F	TxPDO3*, Nd.15	932	0x3A4	TxPDO3*, Nd.36	953	0x3B9	TxPDO3*, Nd.57
912	0x390	TxPDO3*, Nd.16	933	0x3A5	TxPDO3*, Nd.37	954	0x3BA	TxPDO3*, Nd.58
913	0x391	TxPDO3*, Nd.17	934	0x3A6	TxPDO3*, Nd.38	955	0x3BB	TxPDO3*, Nd.59
914	0x392	TxPDO3*, Nd.18	935	0x3A7	TxPDO3*, Nd.39	956	0x3BC	TxPDO3*, Nd.60
915	0x393	TxPDO3*, Nd.19	936	0x3A8	TxPDO3*, Nd.40	957	0x3BD	TxPDO3*, Nd.61
916	0x394	TxPDO3*, Nd.20	937	0x3A9	TxPDO3*, Nd.41	958	0x3BE	TxPDO3*, Nd.62
917	0x395	TxPDO3*, Nd.21	938	0x3AA	TxPDO3*, Nd.42	959	0x3BF	TxPDO3*, Nd.63

dec	hex	Telegram type	dec	hex	Telegram type	dec	hex	Telegram type
1025	0x401	RxPDO3*, Nd.1	1046	0x416	RxPDO3*, Nd.22	1067	0x42B	RxPDO3*, Nd.43
1026	0x402	RxPDO3*, Nd.2	1047	0x417	RxPDO3*, Nd.23	1068	0x42C	RxPDO3*, Nd.44
1027	0x403	RxPDO3*, Nd.3	1048	0x418	RxPDO3*, Nd.24	1069	0x42D	RxPDO3*, Nd.45
1028	0x404	RxPDO3*, Nd.4	1049	0x419	RxPDO3*, Nd.25	1070	0x42E	RxPDO3*, Nd.46
1029	0x405	RxPDO3*, Nd.5	1050	0x41A	RxPDO3*, Nd.26	1071	0x42F	RxPDO3*, Nd.47
1030	0x406	RxPDO3*, Nd.6	1051	0x41B	RxPDO3*, Nd.27	1072	0x430	RxPDO3*, Nd.48
1031	0x407	RxPDO3*, Nd.7	1052	0x41C	RxPDO3*, Nd.28	1073	0x431	RxPDO3*, Nd.49
1032	0x408	RxPDO3*, Nd.8	1053	0x41D	RxPDO3*, Nd.29	1074	0x432	RxPDO3*, Nd.50
1033	0x409	RxPDO3*, Nd.9	1054	0x41E	RxPDO3*, Nd.30	1075	0x433	RxPDO3*, Nd.51
1034	0x40A	RxPDO3*, Nd.10	1055	0x41F	RxPDO3*, Nd.31	1076	0x434	RxPDO3*, Nd.52
1035	0x40B	RxPDO3*, Nd.11	1056	0x420	RxPDO3*, Nd.32	1077	0x435	RxPDO3*, Nd.53
1036	0x40C	RxPDO3*, Nd.12	1057	0x421	RxPDO3*, Nd.33	1078	0x436	RxPDO3*, Nd.54
1037	0x40D	RxPDO3*, Nd.13	1058	0x422	RxPDO3*, Nd.34	1079	0x437	RxPDO3*, Nd.55
1038	0x40E	RxPDO3*, Nd.14	1059	0x423	RxPDO3*, Nd.35	1080	0x438	RxPDO3*, Nd.56
1039	0x40F	RxPDO3*, Nd.15	1060	0x424	RxPDO3*, Nd.36	1081	0x439	RxPDO3*, Nd.57
1040	0x410	RxPDO3*, Nd.16	1061	0x425	RxPDO3*, Nd.37	1082	0x43A	RxPDO3*, Nd.58
1041	0x411	RxPDO3*, Nd.17	1062	0x426	RxPDO3*, Nd.38	1083	0x43B	RxPDO3*, Nd.59
1042	0x412	RxPDO3*, Nd.18	1063	0x427	RxPDO3*, Nd.39	1084	0x43C	RxPDO3*, Nd.60
1043	0x413	RxPDO3*, Nd.19	1064	0x428	RxPDO3*, Nd.40	1085	0x43D	RxPDO3*, Nd.61
1044	0x414	RxPDO3*, Nd.20	1065	0x429	RxPDO3*, Nd.41	1086	0x43E	RxPDO3*, Nd.62
1045	0x415	RxPDO3*, Nd.21	1066	0x42A	RxPDO3*, Nd.42	1087	0x43F	RxPDO3*, Nd.63

dec	hex	Telegram type	dec	hex	Telegram type	dec	hex	Telegram type
1153	0x481	TxPDO4*, Nd.1	1174	0x496	TxPDO4*, Nd.22	1195	0x4AB	TxPDO4*, Nd.43
1154	0x482	TxPDO4*, Nd.2	1175	0x497	TxPDO4*, Nd.23	1196	0x4AC	TxPDO4*, Nd.44
1155	0x483	TxPDO4*, Nd.3	1176	0x498	TxPDO4*, Nd.24	1197	0x4AD	TxPDO4*, Nd.45
1156	0x484	TxPDO4*, Nd.4	1177	0x499	TxPDO4*, Nd.25	1198	0x4AE	TxPDO4*, Nd.46
1157	0x485	TxPDO4*, Nd.5	1178	0x49A	TxPDO4*, Nd.26	1199	0x4AF	TxPDO4*, Nd.47
1158	0x486	TxPDO4*, Nd.6	1179	0x49B	TxPDO4*, Nd.27	1200	0x4B0	TxPDO4*, Nd.48
1159	0x487	TxPDO4*, Nd.7	1180	0x49C	TxPDO4*, Nd.28	1201	0x4B1	TxPDO4*, Nd.49
1160	0x488	TxPDO4*, Nd.8	1181	0x49D	TxPDO4*, Nd.29	1202	0x4B2	TxPDO4*, Nd.50
1161	0x489	TxPDO4*, Nd.9	1182	0x49E	TxPDO4*, Nd.30	1203	0x4B3	TxPDO4*, Nd.51
1162	0x48A	TxPDO4*, Nd.10	1183	0x49F	TxPDO4*, Nd.31	1204	0x4B4	TxPDO4*, Nd.52
1163	0x48B	TxPDO4*, Nd.11	1184	0x4A0	TxPDO4*, Nd.32	1205	0x4B5	TxPDO4*, Nd.53
1164	0x48C	TxPDO4*, Nd.12	1185	0x4A1	TxPDO4*, Nd.33	1206	0x4B6	TxPDO4*, Nd.54
1165	0x48D	TxPDO4*, Nd.13	1186	0x4A2	TxPDO4*, Nd.34	1207	0x4B7	TxPDO4*, Nd.55
1166	0x48E	TxPDO4*, Nd.14	1187	0x4A3	TxPDO4*, Nd.35	1208	0x4B8	TxPDO4*, Nd.56
1167	0x48F	TxPDO4*, Nd.15	1188	0x4A4	TxPDO4*, Nd.36	1209	0x4B9	TxPDO4*, Nd.57
1168	0x490	TxPDO4*, Nd.16	1189	0x4A5	TxPDO4*, Nd.37	1210	0x4BA	TxPDO4*, Nd.58
1169	0x491	TxPDO4*, Nd.17	1190	0x4A6	TxPDO4*, Nd.48	1211	0x4BB	TxPDO4*, Nd.59
1170	0x492	TxPDO4*, Nd.18	1191	0x4A7	TxPDO4*, Nd.49	1212	0x4BC	TxPDO4*, Nd.60
1171	0x493	TxPDO4*, Nd.19	1192	0x4A8	TxPDO4*, Nd.40	1213	0x4BD	TxPDO4*, Nd.61
1172	0x494	TxPDO4*, Nd.20	1193	0x4A9	TxPDO4*, Nd.41	1214	0x4BE	TxPDO4*, Nd.62
1173	0x495	TxPDO4*, Nd.21	1194	0x4AA	TxPDO4*, Nd.42	1215	0x4BF	TxPDO4*, Nd.63

dec	hex	Telegram type	dec	hex	Telegram type	dec	hex	Telegram type
1281	0x501	RxPDO4*, Nd.1	1302	0x516	RxPDO4*, Nd.22	1323	0x52B	RxPDO4*, Nd.43
1282	0x502	RxPDO4*, Nd.2	1303	0x517	RxPDO4*, Nd.23	1324	0x52C	RxPDO4*, Nd.44
1283	0x503	RxPDO4*, Nd.3	1304	0x518	RxPDO4*, Nd.24	1325	0x52D	RxPDO4*, Nd.45
1284	0x504	RxPDO4*, Nd.4	1305	0x519	RxPDO4*, Nd.25	1326	0x52E	RxPDO4*, Nd.46
1285	0x505	RxPDO4*, Nd.5	1306	0x51A	RxPDO4*, Nd.26	1327	0x52F	RxPDO4*, Nd.47
1286	0x506	RxPDO4*, Nd.6	1307	0x51B	RxPDO4*, Nd.27	1328	0x530	RxPDO4*, Nd.48
1287	0x507	RxPDO4*, Nd.7	1308	0x51C	RxPDO4*, Nd.28	1329	0x531	RxPDO4*, Nd.49
1288	0x508	RxPDO4*, Nd.8	1309	0x51D	RxPDO4*, Nd.29	1330	0x532	RxPDO4*, Nd.50
1289	0x509	RxPDO4*, Nd.9	1310	0x51E	RxPDO4*, Nd.30	1331	0x533	RxPDO4*, Nd.51
1290	0x50A	RxPDO4*, Nd.10	1311	0x51F	RxPDO4*, Nd.31	1332	0x534	RxPDO4*, Nd.52
1291	0x50B	RxPDO4*, Nd.11	1312	0x520	RxPDO4*, Nd.32	1333	0x535	RxPDO4*, Nd.53
1292	0x50C	RxPDO4*, Nd.12	1313	0x521	RxPDO4*, Nd.33	1334	0x536	RxPDO4*, Nd.54
1293	0x50D	RxPDO4*, Nd.13	1314	0x522	RxPDO4*, Nd.34	1335	0x537	RxPDO4*, Nd.55
1294	0x50E	RxPDO4*, Nd.14	1315	0x523	RxPDO4*, Nd.35	1336	0x538	RxPDO4*, Nd.56
1295	0x50F	RxPDO4*, Nd.15	1316	0x524	RxPDO4*, Nd.36	1337	0x539	RxPDO4*, Nd.57
1296	0x510	RxPDO4*, Nd.16	1317	0x525	RxPDO4*, Nd.37	1338	0x53A	RxPDO4*, Nd.58
1297	0x511	RxPDO4*, Nd.17	1318	0x526	RxPDO4*, Nd.38	1339	0x53B	RxPDO4*, Nd.59
1298	0x512	RxPDO4*, Nd.18	1319	0x527	RxPDO4*, Nd.39	1340	0x53C	RxPDO4*, Nd.60
1299	0x513	RxPDO4*, Nd.19	1320	0x528	RxPDO4*, Nd.40	1341	0x53D	RxPDO4*, Nd.61
1300	0x514	RxPDO4*, Nd.20	1321	0x529	RxPDO4*, Nd.41	1342	0x53E	RxPDO4*, Nd.62
1301	0x515	RxPDO4*, Nd.21	1322	0x52A	RxPDO4*, Nd.42	1343	0x53F	RxPDO4*, Nd.63

dec	hex	Telegram type	dec	hex	Telegram type	dec	hex	Telegram type
1665	0x681	TxPDO5*, Nd.1	1686	0x696	TxPDO5*, Nd.22	1707	0x6AB	TxPDO5*, Nd.43
1666	0x682	TxPDO5*, Nd.2	1687	0x697	TxPDO5*, Nd.23	1708	0x6AC	TxPDO5*, Nd.44
1667	0x683	TxPDO5*, Nd.3	1688	0x698	TxPDO5*, Nd.24	1709	0x6AD	TxPDO5*, Nd.45
1668	0x684	TxPDO5*, Nd.4	1689	0x699	TxPDO5*, Nd.25	1710	0x6AE	TxPDO5*, Nd.46
1669	0x685	TxPDO5*, Nd.5	1690	0x69A	TxPDO5*, Nd.26	1711	0x6AF	TxPDO5*, Nd.47
1670	0x686	TxPDO5*, Nd.6	1691	0x69B	TxPDO5*, Nd.27	1712	0x6B0	TxPDO5*, Nd.48
1671	0x687	TxPDO5*, Nd.7	1692	0x69C	TxPDO5*, Nd.28	1713	0x6B1	TxPDO5*, Nd.49
1672	0x688	TxPDO5*, Nd.8	1693	0x69D	TxPDO5*, Nd.29	1714	0x6B2	TxPDO5*, Nd.50
1673	0x689	TxPDO5*, Nd.9	1694	0x69E	TxPDO5*, Nd.30	1715	0x6B3	TxPDO5*, Nd.51
1674	0x68A	TxPDO5*, Nd.10	1695	0x69F	TxPDO5*, Nd.31	1716	0x6B4	TxPDO5*, Nd.52
1675	0x68B	TxPDO5*, Nd.11	1696	0x6A0	TxPDO5*, Nd.32	1717	0x6B5	TxPDO5*, Nd.53
1676	0x68C	TxPDO5*, Nd.12	1697	0x6A1	TxPDO5*, Nd.33	1718	0x6B6	TxPDO5*, Nd.54
1677	0x68D	TxPDO5*, Nd.13	1698	0x6A2	TxPDO5*, Nd.34	1719	0x6B7	TxPDO5*, Nd.55
1678	0x68E	TxPDO5*, Nd.14	1699	0x6A3	TxPDO5*, Nd.35	1720	0x6B8	TxPDO5*, Nd.56
1679	0x68F	TxPDO5*, Nd.15	1700	0x6A4	TxPDO5*, Nd.36	1721	0x6B9	TxPDO5*, Nd.57
1680	0x690	TxPDO5*, Nd.16	1701	0x6A5	TxPDO5*, Nd.37	1722	0x6BA	TxPDO5*, Nd.58
1681	0x691	TxPDO5*, Nd.17	1702	0x6A6	TxPDO5*, Nd.38	1723	0x6BB	TxPDO5*, Nd.59
1682	0x692	TxPDO5*, Nd.18	1703	0x6A7	TxPDO5*, Nd.39	1724	0x6BC	TxPDO5*, Nd.60
1683	0x693	TxPDO5*, Nd.19	1704	0x6A8	TxPDO5*, Nd.40	1725	0x6BD	TxPDO5*, Nd.61
1684	0x694	TxPDO5*, Nd.20	1705	0x6A9	TxPDO5*, Nd.41	1726	0x6BE	TxPDO5*, Nd.62
1685	0x695	TxPDO5*, Nd.21	1706	0x6AA	TxPDO5*, Nd.42	1727	0x6BF	TxPDO5*, Nd.63

dec	hex	Telegram type	dec	hex	Telegram type	dec	hex	Telegram type
1921	0x781	RxPDO5*, Nd.1	1942	0x796	RxPDO5*, Nd.22	1963	0x7AB	RxPDO5*, Nd.43
1922	0x782	RxPDO5*, Nd.2	1943	0x797	RxPDO5*, Nd.23	1964	0x7AC	RxPDO5*, Nd.44
1923	0x783	RxPDO5*, Nd.3	1944	0x798	RxPDO5*, Nd.24	1965	0x7AD	RxPDO5*, Nd.45
1924	0x784	RxPDO5*, Nd.4	1945	0x799	RxPDO5*, Nd.25	1966	0x7AE	RxPDO5*, Nd.46
1925	0x785	RxPDO5*, Nd.5	1946	0x79A	RxPDO5*, Nd.26	1967	0x7AF	RxPDO5*, Nd.47
1926	0x786	RxPDO5*, Nd.6	1947	0x79B	RxPDO5*, Nd.27	1968	0x7B0	RxPDO5*, Nd.48
1927	0x787	RxPDO5*, Nd.7	1948	0x79C	RxPDO5*, Nd.28	1969	0x7B1	RxPDO5*, Nd.49
1928	0x788	RxPDO5*, Nd.8	1949	0x79D	RxPDO5*, Nd.29	1970	0x7B2	RxPDO5*, Nd.50
1929	0x789	RxPDO5*, Nd.9	1950	0x79E	RxPDO5*, Nd.30	1971	0x7B3	RxPDO5*, Nd.51
1930	0x78A	RxPDO5*, Nd.10	1951	0x79F	RxPDO5*, Nd.31	1972	0x7B4	RxPDO5*, Nd.52
1931	0x78B	RxPDO5*, Nd.11	1952	0x7A0	RxPDO5*, Nd.32	1973	0x7B5	RxPDO5*, Nd.53
1932	0x78C	RxPDO5*, Nd.12	1953	0x7A1	RxPDO5*, Nd.33	1974	0x7B6	RxPDO5*, Nd.54
1933	0x78D	RxPDO5*, Nd.13	1954	0x7A2	RxPDO5*, Nd.34	1975	0x7B7	RxPDO5*, Nd.55
1934	0x78E	RxPDO5*, Nd.14	1955	0x7A3	RxPDO5*, Nd.35	1976	0x7B8	RxPDO5*, Nd.56
1935	0x78F	RxPDO5*, Nd.15	1956	0x7A4	RxPDO5*, Nd.36	1977	0x7B9	RxPDO5*, Nd.57
1936	0x790	RxPDO5*, Nd.16	1957	0x7A5	RxPDO5*, Nd.37	1978	0x7BA	RxPDO5*, Nd.58
1937	0x791	RxPDO5*, Nd.17	1958	0x7A6	RxPDO5*, Nd.38	1979	0x7BB	RxPDO5*, Nd.59
1938	0x792	RxPDO5*, Nd.18	1959	0x7A7	RxPDO5*, Nd.39	1980	0x7BC	RxPDO5*, Nd.60
1939	0x793	RxPDO5*, Nd.19	1960	0x7A8	RxPDO5*, Nd.40	1981	0x7BD	RxPDO5*, Nd.61
1940	0x794	RxPDO5*, Nd.20	1961	0x7A9	RxPDO5*, Nd.41	1982	0x7BE	RxPDO5*, Nd.62
1941	0x795	RxPDO5*, Nd.21	1962	0x7AA	RxPDO5*, Nd.42	1983	0x7BF	RxPDO5*, Nd.63

dec	hex	Telegram type	dec	hex	Telegram type	dec	hex	Telegram type
449	0x1C1	TxPDO6*, Nd.1	470	0x1D6	TxPDO6*, Nd.22	491	0x1EB	TxPDO6*, Nd.43
450	0x1C2	TxPDO6*, Nd.2	471	0x1D7	TxPDO6*, Nd.23	492	0x1EC	TxPDO6*, Nd.44
451	0x1C3	TxPDO6*, Nd.3	472	0x1D8	TxPDO6*, Nd.24	493	0x1ED	TxPDO6*, Nd.45
452	0x1C4	TxPDO6*, Nd.4	473	0x1D9	TxPDO6*, Nd.25	494	0x1EE	TxPDO6*, Nd.46
453	0x1C5	TxPDO6*, Nd.5	474	0x1DA	TxPDO6*, Nd.26	495	0x1EF	TxPDO6*, Nd.47
454	0x1C6	TxPDO6*, Nd.6	475	0x1DB	TxPDO6*, Nd.27	496	0x1F0	TxPDO6*, Nd.48
455	0x1C7	TxPDO6*, Nd.7	476	0x1DC	TxPDO6*, Nd.28	497	0x1F1	TxPDO6*, Nd.49
456	0x1C8	TxPDO6*, Nd.8	477	0x1DD	TxPDO6*, Nd.29	498	0x1F2	TxPDO6*, Nd.50
457	0x1C9	TxPDO6*, Nd.9	478	0x1DE	TxPDO6*, Nd.30	499	0x1F3	TxPDO6*, Nd.51
458	0x1CA	TxPDO6*, Nd.10	479	0x1DF	TxPDO6*, Nd.31	500	0x1F4	TxPDO6*, Nd.52
459	0x1CB	TxPDO6*, Nd.11	480	0x1E0	TxPDO6*, Nd.32	501	0x1F5	TxPDO6*, Nd.53
460	0x1CC	TxPDO6*, Nd.12	481	0x1E1	TxPDO6*, Nd.33	502	0x1F6	TxPDO6*, Nd.54
461	0x1CD	TxPDO6*, Nd.13	482	0x1E2	TxPDO6*, Nd.34	503	0x1F7	TxPDO6*, Nd.55
462	0x1CE	TxPDO6*, Nd.14	483	0x1E3	TxPDO6*, Nd.35	504	0x1F8	TxPDO6*, Nd.56
463	0x1CF	TxPDO6*, Nd.15	484	0x1E4	TxPDO6*, Nd.36	505	0x1F9	TxPDO6*, Nd.57
464	0x1D0	TxPDO6*, Nd.16	485	0x1E5	TxPDO6*, Nd.37	506	0x1FA	TxPDO6*, Nd.58
465	0x1D1	TxPDO6*, Nd.17	486	0x1E6	TxPDO6*, Nd.38	507	0x1FB	TxPDO6*, Nd.59
466	0x1D2	TxPDO6*, Nd.18	487	0x1E7	TxPDO6*, Nd.39	508	0x1FC	TxPDO6*, Nd.60
467	0x1D3	TxPDO6*, Nd.19	488	0x1E8	TxPDO6*, Nd.40	509	0x1FD	TxPDO6*, Nd.61
468	0x1D4	TxPDO6*, Nd.20	489	0x1E9	TxPDO6*, Nd.41	510	0x1FE	TxPDO6*, Nd.62
469	0x1D5	TxPDO6*, Nd.21	490	0x1EA	TxPDO6*, Nd.42	511	0x1FF	TxPDO6*, Nd.63

dec	hex	Telegram type	dec	hex	Telegram type	dec	hex	Telegram type
577	0x241	RxPDO6*, Nd.1	598	0x256	RxPDO6*, Nd.22	619	0x26B	RxPDO6* Nd.43
578	0x242	RxPDO6*, Nd.2	599	0x257	RxPDO6*, Nd.23	620	0x26C	RxPDO6, Nd.44
579	0x243	RxPDO6*, Nd.3	600	0x258	RxPDO6*, Nd.24	621	0x26D	RxPDO6*, Nd.45
580	0x244	RxPDO6*, Nd.4	601	0x259	RxPDO6*, Nd.25	622	0x26E	RxPDO6*, Nd.46
581	0x245	RxPDO6*, Nd.5	602	0x25A	RxPDO6*, Nd.26	623	0x26F	RxPDO6*, Nd.47
582	0x246	RxPDO6*, Nd.6	603	0x25B	RxPDO6*, Nd.27	624	0x270	RxPDO6*, Nd.48
583	0x247	RxPDO6*, Nd.7	604	0x25C	RxPDO6*, Nd.28	625	0x271	RxPDO6*, Nd.49
584	0x248	RxPDO6*, Nd.8	605	0x25D	RxPDO6*, Nd.29	626	0x272	RxPDO6*, Nd.50
585	0x249	RxPDO6*, Nd.9	606	0x25E	RxPDO6*, Nd.30	627	0x273	RxPDO6*, Nd.51
586	0x24A	RxPDO6*, Nd.10	607	0x25F	RxPDO6*, Nd.31	628	0x274	RxPDO6*, Nd.52
587	0x24B	RxPDO6*, Nd.11	608	0x260	RxPDO6*, Nd.32	629	0x275	RxPDO6*, Nd.53
588	0x24C	RxPDO6*, Nd.12	609	0x261	RxPDO6*, Nd.33	630	0x276	RxPDO6*, Nd.54
589	0x24D	RxPDO6*, Nd.13	610	0x262	RxPDO6*, Nd.34	631	0x277	RxPDO6*, Nd.55
590	0x24E	RxPDO6*, Nd.14	611	0x263	RxPDO6*, Nd.35	632	0x278	RxPDO6*, Nd.56
591	0x24F	RxPDO6*, Nd.15	612	0x264	RxPDO6*, Nd.36	633	0x279	RxPDO6*, Nd.57
592	0x250	RxPDO6*, Nd.16	613	0x265	RxPDO6*, Nd.3	634	0x27A	RxPDO6*, Nd.58
593	0x251	RxPDO6*, Nd.17	614	0x266	RxPDO6*, Nd.8	635	0x27B	RxPDO6*, Nd.59
594	0x252	RxPDO6*, Nd.18	615	0x267	RxPDO6*, Nd39	636	0x27C	RxPDO6*, Nd.60
595	0x253	RxPDO6*, Nd.19	616	0x268	RxPDO6*, N.40	637	0x27D	RxPDO6*, Nd.61
596	0x254	RxPDO6*, Nd.20	617	0x269	RxPDO6*, d.41	638	0x27E	RxPDO6*, Nd.62
597	0x255	RxPDO6*, Nd.21	618	0x26A	RxPDO6*,Nd.42	639	0x27F	RxPDO6*, Nd.63

dec	hex	Telegram type	dec	hex	Telegram type	dec	hex	Telegram type
705	0x2C1	TxPDO7*, Nd.1	726	0x2D6	TxPDO7*, Nd.22	747	0x2EB	TxPDO7*, Nd.43
706	0x2C2	TxPDO7*, Nd.2	727	0x2D7	TxPDO7*, Nd.23	748	0x2EC	TxPDO7*, Nd.44
707	0x2C3	TxPDO7*, Nd.3	728	0x2D8	TxPDO7*, Nd.24	749	0x2ED	TxPDO7*, Nd.45
708	0x2C4	TxPDO7*, Nd.4	729	0x2D9	TxPDO7*, Nd.25	750	0x2EE	TxPDO7*, Nd.46
709	0x2C5	TxPDO7*, Nd.5	730	0x2DA	TxPDO7*, Nd.26	751	0x2EF	TxPDO7*, Nd.47
710	0x2C6	TxPDO7*, Nd.6	731	0x2DB	TxPDO7*, Nd.27	752	0x2F0	TxPDO7*, Nd.48
711	0x2C7	TxPDO7*, Nd.7	732	0x2DC	TxPDO7*, Nd.28	753	0x2F1	TxPDO7*, Nd.49
712	0x2C8	TxPDO7*, Nd.8	733	0x2DD	TxPDO7*, Nd.29	754	0x2F2	TxPDO7*, Nd.50
713	0x2C9	TxPDO7*, Nd.9	734	0x2DE	TxPDO7*, Nd.30	755	0x2F3	TxPDO7*, Nd.51
714	0x2CA	TxPDO7*, Nd.10	735	0x2DF	TxPDO7*, Nd.31	756	0x2F4	TxPDO7*, Nd.52
715	0x2CB	TxPDO7*, Nd.11	736	0x2E0	TxPDO7*, Nd.32	757	0x2F5	TxPDO7*, Nd.53
716	0x2CC	TxPDO7*, Nd.12	737	0x2E1	TxPDO7*, Nd.33	758	0x2F6	TxPDO7*, Nd.54
717	0x2CD	TxPDO7*, Nd.13	738	0x2E2	TxPDO7*, Nd.34	759	0x2F7	TxPDO7*, Nd.55
718	0x2CE	TxPDO7*, Nd.14	739	0x2E3	TxPDO7*, Nd.35	760	0x2F8	TxPDO7*, Nd.56
719	0x2CF	TxPDO7*, Nd.15	740	0x2E4	TxPDO7*, Nd.36	761	0x2F9	TxPDO7*, Nd.57
720	0x2D0	TxPDO7*, Nd.16	741	0x2E5	TxPDO7*, Nd.37	762	0x2FA	TxPDO7*, Nd.58
721	0x2D1	TxPDO7*, Nd.17	742	0x2E6	TxPDO7*, Nd.38	763	0x2FB	TxPDO7*, Nd.59
722	0x2D2	TxPDO7*, Nd.18	743	0x2E7	TxPDO7*, Nd.39	764	0x2FC	TxPDO7*, Nd.60
723	0x2D3	TxPDO7*, Nd.19	744	0x2E8	TxPDO7*, Nd.40	765	0x2FD	TxPDO7*, Nd.61
724	0x2D4	TxPDO7*, Nd.20	745	0x2E9	TxPDO7*, Nd.41	766	0x2FE	TxPDO7*, Nd.62
725	0x2D5	TxPDO7*, Nd.21	746	0x2EA	TxPDO7*, Nd.42	767	0x2FF	TxPDO7*, Nd.63

dec	hex	Telegram type	dec	hex	Telegram type	dec	hex	Telegram type
833	0x341	RxPDO7*, Nd.1	854	0x356	RxPDO7*, Nd.22	875	0x36B	RxPDO7*, Nd.43
834	0x342	RxPDO7*, Nd.2	855	0x357	RxPDO7*, Nd.23	876	0x36C	RxPDO7*, Nd.44
835	0x343	RxPDO7*, Nd.3	856	0x358	RxPDO7*, Nd.24	877	0x36D	RxPDO7*, Nd.45
836	0x344	RxPDO7*, Nd.4	857	0x359	RxPDO7*, Nd.25	878	0x36E	RxPDO7*, Nd.46
837	0x345	RxPDO7*, Nd.5	858	0x35A	RxPDO7*, Nd.26	879	0x36F	RxPDO7*, Nd.47
838	0x346	RxPDO7*, Nd.6	859	0x35B	RxPDO7*, Nd.27	880	0x370	RxPDO7*, Nd.48
839	0x347	RxPDO7*, Nd.7	860	0x35C	RxPDO7*, Nd.28	881	0x371	RxPDO7*, Nd.49
840	0x348	RxPDO7*, Nd.8	861	0x35D	RxPDO7*, Nd.29	882	0x372	RxPDO7*, Nd.50
841	0x349	RxPDO7*, Nd.9	862	0x35E	RxPDO7*, Nd.30	883	0x373	RxPDO7*, Nd.51
842	0x34A	RxPDO7*, Nd.10	863	0x35F	RxPDO7*, Nd.31	884	0x374	RxPDO7*, Nd.52
843	0x34B	RxPDO7*, Nd.11	864	0x360	RxPDO7*, Nd.32	885	0x375	RxPDO7*, Nd.53
844	0x34C	RxPDO7*, Nd.12	865	0x361	RxPDO7*, Nd.33	886	0x376	RxPDO7*, Nd.54
845	0x34D	RxPDO7*, Nd.13	866	0x362	RxPDO7*, Nd.34	887	0x377	RxPDO7*, Nd.55
846	0x34E	RxPDO7*, Nd.14	867	0x363	RxPDO7*, Nd.35	888	0x378	RxPDO7*, Nd.56
847	0x34F	RxPDO7*, Nd.15	868	0x364	RxPDO7*, Nd.36	889	0x379	RxPDO7*, Nd.57
848	0x350	RxPDO7*, Nd.16	869	0x365	RxPDO7*, Nd.37	890	0x37A	RxPDO7*, Nd.58
849	0x351	RxPDO7*, Nd.17	870	0x366	RxPDO7*, Nd.38	891	0x37B	RxPDO7*, Nd.59
850	0x352	RxPDO7*, Nd.18	871	0x367	RxPDO7*, Nd.39	892	0x37C	RxPDO7*, Nd.60
851	0x353	RxPDO7*, Nd.19	872	0x368	RxPDO7*, Nd.40	893	0x37D	RxPDO7*, Nd.61
852	0x354	RxPDO7*, Nd.20	873	0x369	RxPDO7*, Nd.41	894	0x37E	RxPDO7*, Nd.62
853	0x355	RxPDO7*, Nd.21	874	0x36A	RxPDO7*, Nd.42	895	0x37F	RxPDO7*, Nd.63

dec	hex	Telegram type	dec	hex	Telegram type	dec	hex	Telegram type
961	0x3C1	TxPDO8*, Nd.1	982	0x3D6	TxPDO8*, Nd.22	1003	0x3EB	TxPDO8*, Nd.43
962	0x3C2	TxPDO8*, Nd.2	983	0x3D7	TxPDO8*, Nd.23	1004	0x3EC	TxPDO8*, Nd.44
963	0x3C3	TxPDO8*, Nd.3	984	0x3D8	TxPDO8*, Nd.24	1005	0x3ED	TxPDO8*, Nd.45
964	0x3C4	TxPDO8*, Nd.4	985	0x3D9	TxPDO8*, Nd.25	1006	0x3EE	TxPDO8*, Nd.46
965	0x3C5	TxPDO8*, Nd.5	986	0x3DA	TxPDO8*, Nd.26	1007	0x3EF	TxPDO8*, Nd.47
966	0x3C6	TxPDO8*, Nd.6	987	0x3DB	TxPDO8*, Nd.27	1008	0x3F0	TxPDO8*, Nd.48
967	0x3C7	TxPDO8*, Nd.7	988	0x3DC	TxPDO8*, Nd.28	1009	0x3F1	TxPDO8*, Nd.49
968	0x3C8	TxPDO8*, Nd.8	989	0x3DD	TxPDO8*, Nd.29	1010	0x3F2	TxPDO8*, Nd.50
969	0x3C9	TxPDO8*, Nd.9	990	0x3DE	TxPDO8*, Nd.30	1011	0x3F3	TxPDO8*, Nd.51
970	0x3CA	TxPDO8*, Nd.10	991	0x3DF	TxPDO8*, Nd.31	1012	0x3F4	TxPDO8*, Nd.52
971	0x3CB	TxPDO8*, Nd.11	992	0x3E0	TxPDO8*, Nd.32	1013	0x3F5	TxPDO8*, Nd.53
972	0x3CC	TxPDO8*, Nd.12	993	0x3E1	TxPDO8*, Nd.33	1014	0x3F6	TxPDO8*, Nd.54
973	0x3CD	TxPDO8*, Nd.13	994	0x3E2	TxPDO8*, Nd.34	1015	0x3F7	TxPDO8*, Nd.55
974	0x3CE	TxPDO8*, Nd.14	995	0x3E3	TxPDO8*, Nd.35	1016	0x3F8	TxPDO8*, Nd.56
975	0x3CF	TxPDO8*, Nd.15	996	0x3E4	TxPDO8*, Nd.36	1017	0x3F9	TxPDO8*, Nd.57
976	0x3D0	TxPDO8*, Nd.16	997	0x3E5	TxPDO8*, Nd.37	1018	0x3FA	TxPDO8*, Nd.58
977	0x3D1	TxPDO8*, Nd.17	998	0x3E6	TxPDO8*, Nd.38	1019	0x3FB	TxPDO8*, Nd.59
978	0x3D2	TxPDO8*, Nd.18	999	0x3E7	TxPDO8*, Nd.39	1020	0x3FC	TxPDO8*, Nd.60
979	0x3D3	TxPDO8*, Nd.19	1000	0x3E8	TxPDO8*, Nd.40	1021	0x3FD	TxPDO8*, Nd.61
980	0x3D4	TxPDO8*, Nd.20	1001	0x3E9	TxPDO8*, Nd.41	1022	0x3FE	TxPDO8*, Nd.62
981	0x3D5	TxPDO8*, Nd.21	1002	0x3EA	TxPDO8*, Nd.42	1023	0x3FF	TxPDO8*, Nd.63

dec	hex	Telegram type	dec	hex	Telegram type	dec	hex	Telegram type
1089	0x441	RxPDO8*, Nd.1	1110	0x456	RxPDO8*, Nd.22	1131	0x46B	RxPDO8*, Nd.43
1090	0x442	RxPDO8*, Nd.2	1111	0x457	RxPDO8*, Nd.23	1132	0x46C	RxPDO8*, Nd.44
1091	0x443	RxPDO8*, Nd.3	1112	0x458	RxPDO8*, Nd.24	1133	0x46D	RxPDO8*, Nd.45
1092	0x444	RxPDO8*, Nd.4	1113	0x459	RxPDO8*, Nd.25	1134	0x46E	RxPDO8*, Nd.46
1093	0x445	RxPDO8*, Nd.5	1114	0x45A	RxPDO8*, Nd.26	1135	0x46F	RxPDO8*, Nd.47
1094	0x446	RxPDO8*, Nd.6	1115	0x45B	RxPDO8*, Nd.27	1136	0x470	RxPDO8*, Nd.48
1095	0x447	RxPDO8*, Nd.7	1116	0x45C	RxPDO8*, Nd.28	1137	0x471	RxPDO8*, Nd.49
1096	0x448	RxPDO8*, Nd.8	1117	0x45D	RxPDO8*, Nd.29	1138	0x472	RxPDO8*, Nd.50
1097	0x449	RxPDO8*, Nd.9	1118	0x45E	RxPDO8*, Nd.30	1139	0x473	RxPDO8*, Nd.51
1098	0x44A	RxPDO8*, Nd.10	1119	0x45F	RxPDO8*, Nd.31	1140	0x474	RxPDO8*, Nd.52
1099	0x44B	RxPDO8*, Nd.11	1120	0x460	RxPDO8*, Nd.32	1141	0x475	RxPDO8*, Nd.53
1100	0x44C	RxPDO8*, Nd.12	1121	0x461	RxPDO8*, Nd.33	1142	0x476	RxPDO8*, Nd.54
1101	0x44D	RxPDO8*, Nd.13	1122	0x462	RxPDO8*, Nd.34	1143	0x477	RxPDO8*, Nd.55
1102	0x44E	RxPDO8*, Nd.14	1123	0x463	RxPDO8*, Nd.35	1144	0x478	RxPDO8*, Nd.56
1103	0x44F	RxPDO8*, Nd.15	1124	0x464	RxPDO8*, Nd.36	1145	0x479	RxPDO8*, Nd.57
1104	0x450	RxPDO8*, Nd.16	1125	0x465	RxPDO8*, Nd.37	1146	0x47A	RxPDO8*, Nd.58
1105	0x451	RxPDO8*, Nd.17	1126	0x466	RxPDO8*, Nd.38	1147	0x47B	RxPDO8*, Nd.59
1106	0x452	RxPDO8*, Nd.18	1127	0x467	RxPDO8*, Nd.39	1148	0x47C	RxPDO8*, Nd.60
1107	0x453	RxPDO8*, Nd.19	1128	0x468	RxPDO8*, Nd.40	1149	0x47D	RxPDO8*, Nd.61
1108	0x454	RxPDO8*, Nd.20	1129	0x469	RxPDO8*, Nd.41	1150	0x47E	RxPDO8*, Nd.62
1109	0x455	RxPDO8*, Nd.21	1130	0x46A	RxPDO8*, Nd.42	1151	0x47F	RxPDO8*, Nd.63

dec	hex	Telegram type	dec	hex	Telegram type	dec	hex	Telegram type
1217	0x4C1	TxPDO9*, Nd.1	1238	0x4D6	TxPDO9*, Nd.22	1259	0x4EB	TxPDO9*, Nd.43
1218	0x4C2	TxPDO9*, Nd.2	1239	0x4D7	TxPDO9*, Nd.23	1260	0x4EC	TxPDO9*, Nd.44
1219	0x4C3	TxPDO9*, Nd.3	1240	0x4D8	TxPDO9*, Nd.24	1261	0x4ED	TxPDO9*, Nd.45
1220	0x4C4	TxPDO9*, Nd.4	1241	0x4D9	TxPDO9*, Nd.25	1262	0x4EE	TxPDO9*, Nd.46
1221	0x4C5	TxPDO9*, Nd.5	1242	0x4DA	TxPDO9*, Nd.26	1263	0x4EF	TxPDO9*, Nd.47
1222	0x4C6	TxPDO9*, Nd.6	1243	0x4DB	TxPDO9*, Nd.27	1264	0x4F0	TxPDO9*, Nd.48
1223	0x4C7	TxPDO9*, Nd.7	1244	0x4DC	TxPDO9*, Nd.28	1265	0x4F1	TxPDO9*, Nd.49
1224	0x4C8	TxPDO9*, Nd.8	1245	0x4DD	TxPDO9*, Nd.29	1266	0x4F2	TxPDO9*, Nd.50
1225	0x4C9	TxPDO9*, Nd.9	1246	0x4DE	TxPDO9*, Nd.30	1267	0x4F3	TxPDO9*, Nd.51
1226	0x4CA	TxPDO9*, Nd.10	1247	0x4DF	TxPDO9*, Nd.31	1268	0x4F4	TxPDO9*, Nd.52
1227	0x4CB	TxPDO9*, Nd.11	1248	0x4E0	TxPDO9*, Nd.32	1269	0x4F5	TxPDO9*, Nd.53
1228	0x4CC	TxPDO9*, Nd.12	1249	0x4E1	TxPDO9*, Nd.33	1270	0x4F6	TxPDO9*, Nd.54
1229	0x4CD	TxPDO9*, Nd.13	1250	0x4E2	TxPDO9*, Nd.34	1271	0x4F7	TxPDO9*, Nd.55
1230	0x4CE	TxPDO9*, Nd.14	1251	0x4E3	TxPDO9*, Nd.35	1272	0x4F8	TxPDO9*, Nd.56
1231	0x4CF	TxPDO9*, Nd.15	1252	0x4E4	TxPDO9*, Nd.36	1273	0x4F9	TxPDO9*, Nd.57
1232	0x4D0	TxPDO9*, Nd.16	1253	0x4E5	TxPDO9*, Nd.37	1274	0x4FA	TxPDO9*, Nd.58
1233	0x4D1	TxPDO9*, Nd.17	1254	0x4E6	TxPDO9*, Nd.38	1275	0x4FB	TxPDO9*, Nd.59
1234	0x4D2	TxPDO9*, Nd.18	1255	0x4E7	TxPDO9*, Nd.39	1276	0x4FC	TxPDO9*, Nd.60
1235	0x4D3	TxPDO9*, Nd.19	1256	0x4E8	TxPDO9*, Nd.40	1277	0x4FD	TxPDO9*, Nd.61
1236	0x4D4	TxPDO9*, Nd.20	1257	0x4E9	TxPDO9*, Nd.41	1278	0x4FE	TxPDO9*, Nd.62
1237	0x4D5	TxPDO9*, Nd.21	1258	0x4EA	TxPDO9*, Nd.42	1279	0x4FF	TxPDO9*, Nd.63

dec	hex	Telegram type	dec	hex	Telegram type	dec	hex	Telegram type
1345	0x541	RxPDO9*, Nd.1	1366	0x556	RxPDO9*, Nd.22	1387	0x56B	RxPDO9*, Nd.43
1346	0x542	RxPDO9*, Nd.2	1367	0x557	RxPDO9*, Nd.23	1388	0x56C	RxPDO9*, Nd.44
1347	0x543	RxPDO9*, Nd.3	1368	0x558	RxPDO9*, Nd.24	1389	0x56D	RxPDO9*, Nd.45
1348	0x544	RxPDO9*, Nd.4	1369	0x559	RxPDO9*, Nd.25	1390	0x56E	RxPDO9*, Nd.46
1349	0x545	RxPDO9*, Nd.5	1370	0x55A	RxPDO9*, Nd.26	1391	0x56F	RxPDO9*, Nd.47
1350	0x546	RxPDO9*, Nd.6	1371	0x55B	RxPDO9*, Nd.27	1392	0x570	RxPDO9*, Nd.48
1351	0x547	RxPDO9*, Nd.7	1372	0x55C	RxPDO9*, Nd.28	1393	0x571	RxPDO9*, Nd.49
1352	0x548	RxPDO9*, Nd.8	1373	0x55D	RxPDO9*, Nd.29	1394	0x572	RxPDO9*, Nd.50
1353	0x549	RxPDO9*, Nd.9	1374	0x55E	RxPDO9*, Nd.30	1395	0x573	RxPDO9*, Nd.51
1354	0x54A	RxPDO9*, Nd.10	1375	0x55F	RxPDO9*, Nd.31	1396	0x574	RxPDO9*, Nd.52
1355	0x54B	RxPDO9*, Nd.11	1376	0x560	RxPDO9*, Nd.32	1397	0x575	RxPDO9*, Nd.53
1356	0x54C	RxPDO9*, Nd.12	1377	0x561	RxPDO9*, Nd.33	1398	0x576	RxPDO9*, Nd.54
1357	0x54D	RxPDO9*, Nd.13	1378	0x562	RxPDO9*, Nd.34	1399	0x577	RxPDO9*, Nd.55
1358	0x54E	RxPDO9*, Nd.14	1379	0x563	RxPDO9*, Nd.35	1400	0x578	RxPDO9*, Nd.56
1359	0x54F	RxPDO9*, Nd.15	1380	0x564	RxPDO9*, Nd.36	1401	0x579	RxPDO9*, Nd.57
1360	0x550	RxPDO9*, Nd.16	1381	0x565	RxPDO9*, Nd.37	1402	0x57A	RxPDO9*, Nd.58
1361	0x551	RxPDO9*, Nd.17	1382	0x566	RxPDO9*, Nd.38	1403	0x57B	RxPDO9*, Nd.59
1362	0x552	RxPDO9*, Nd.18	1383	0x567	RxPDO9*, Nd.39	1404	0x57C	RxPDO9*, Nd.60
1363	0x553	RxPDO9*, Nd.19	1384	0x568	RxPDO9*, Nd.40	1405	0x57D	RxPDO9*, Nd.61
1364	0x554	RxPDO9*, Nd.20	1385	0x569	RxPDO9*, Nd.41	1406	0x57E	RxPDO9*, Nd.62
1365	0x555	RxPDO9*, Nd.21	1386	0x56A	RxPDO9*, Nd.42	1407	0x57F	RxPDO9*, Nd.63

dec	hex	Telegram type	dec	hex	Telegram type	dec	hex	Telegram type
1473	0x5C1	TxPDO10*, Nd.1	1494	0x5D6	TxPDO10*, Nd.22	1515	0x5EB	TxPDO10*, Nd.43
1474	0x5C2	TxPDO10*, Nd.2	1495	0x5D7	TxPDO10*, Nd.23	1516	0x5EC	TxPDO10*, Nd.44
1475	0x5C3	TxPDO10*, Nd.3	1496	0x5D8	TxPDO10*, Nd.24	1517	0x5ED	TxPDO10*, Nd.45
1476	0x5C4	TxPDO10*, Nd.4	1497	0x5D9	TxPDO10*, Nd.25	1518	0x5EE	TxPDO10*, Nd.46
1477	0x5C5	TxPDO10*, Nd.5	1498	0x5DA	TxPDO10*, Nd.26	1519	0x5EF	TxPDO10*, Nd.47
1478	0x5C6	TxPDO10*, Nd.6	1499	0x5DB	TxPDO10*, Nd.27	1520	0x5F0	TxPDO10*, Nd.48
1479	0x5C7	TxPDO10*, Nd.7	1500	0x5DC	TxPDO10*, Nd.28	1521	0x5F1	TxPDO10*, Nd.49
1480	0x5C8	TxPDO10*, Nd.8	1501	0xDE	TxPDO10*, Nd.29	1522	0x5F2	TxPDO10*, Nd.50
1481	0x5C9	TxPDO10*, Nd.9	1502	0x5DE	TxPDO10*, Nd.30	1523	0x5F3	TxPDO10*, Nd.51
1482	0x5CA	TxPDO10*, Nd.10	1503	0x5DF	TxPDO10*, Nd.31	1524	0x5F4	TxPDO10*, Nd.52
1483	0x5CB	TxPDO10*, Nd.11	1504	0x5E0	TxPDO10*, Nd.32	1525	0x5F5	TxPDO10*, Nd.53
1484	0x5CC	TxPDO10*, Nd.12	1505	0x5E1	TxPDO10*, Nd.33	1526	0x5F6	TxPDO10*, Nd.54
1485	0x5CD	TxPDO10*, Nd.13	1506	0x5E2	TxPDO10*, Nd.34	1527	0x5F7	TxPDO10*, Nd.55
1486	0x5CE	TxPDO10*, Nd.14	1507	0x5E3	TxPDO10*, Nd.35	1528	0x5F8	TxPDO10*, Nd.56
1487	0x5CF	TxPDO10*, Nd.15	1508	0x5E4	TxPDO10*, Nd.36	1529	0x5F9	TxPDO10*, Nd.57
1488	0x5D0	TxPDO10*, Nd.16	1509	0x5E5	TxPDO10*, Nd.37	1530	0x5FA	TxPDO10*, Nd.58
1489	0x5D1	TxPDO10*, Nd.17	1510	0x5E6	TxPDO10*, Nd.38	1531	0x5FB	TxPDO10*, Nd.59
1490	0x5D2	TxPDO10*, Nd.18	1511	0x5E7	TxPDO10*, Nd.39	1532	0x5FC	TxPDO10*, Nd.60
1491	0x5D3	TxPDO10*, Nd.19	1512	0x5E8	TxPDO10*, Nd.40	1533	0x5FD	TxPDO10*, Nd.61
1492	0x5D4	TxPDO10*, Nd.20	1513	0x5E9	TxPDO10*, Nd.41	1534	0x5FE	TxPDO10*, Nd.62
1493	0x5D5	TxPDO10*, Nd.21	1514	0x5EA	TxPDO10*, Nd.42	1535	0x5FF	TxPDO10*, Nd.63

dec	hex	Telegram type	dec	hex	Telegram type	dec	hex	Telegram type
1601	0x641	RxPDO10*, Nd.1	1622	0x656	RxPDO10*, Nd.22	1643	0x66B	RxPDO10*, Nd.43
1602	0x642	RxPDO10*, Nd.2	1623	0x657	RxPDO10*, Nd.23	1644	0x66C	RxPDO10*, Nd.44
1603	0x643	RxPDO10*, Nd.3	1624	0x658	RxPDO10*, Nd.24	1645	0x66D	RxPDO10*, Nd.45
1604	0x644	RxPDO10*, Nd.4	1625	0x659	RxPDO10*, Nd.25	1646	0x66E	RxPDO10*, Nd.46
1605	0x645	RxPDO10*, Nd.5	1626	0x65A	RxPDO10*, Nd.26	1647	0x66F	RxPDO10*, Nd.47
1606	0x646	RxPDO10*, Nd.6	1627	0x65B	RxPDO10*, Nd.27	1648	0x670	RxPDO10*, Nd.48
1607	0x647	RxPDO10*, Nd.7	1628	0x65C	RxPDO10*, Nd.28	1649	0x671	RxPDO10*, Nd.49
1608	0x648	RxPDO10*, Nd.8	1629	0x65D	RxPDO10*, Nd.29	1650	0x672	RxPDO10*, Nd.50
1609	0x649	RxPDO10*, Nd.9	1630	0x65E	RxPDO10*, Nd.30	1651	0x673	RxPDO10*, Nd.51
1610	0x64A	RxPDO10*, Nd.10	1631	0x65F	RxPDO10*, Nd.31	1652	0x674	RxPDO10*, Nd.52
1611	0x64B	RxPDO10*, Nd.11	1632	0x660	RxPDO10*, Nd.32	1653	0x675	RxPDO10*, Nd.53
1612	0x64C	RxPDO10*, Nd.12	1633	0x661	RxPDO10*, Nd.33	1654	0x676	RxPDO10*, Nd.54
1613	0x64D	RxPDO10*, Nd.13	1634	0x662	RxPDO10*, Nd.34	1655	0x677	RxPDO10*, Nd.55
1614	0x64E	RxPDO10*, Nd.14	1635	0x663	RxPDO10*, Nd.35	1656	0x678	RxPDO10*, Nd.56
1615	0x64F	RxPDO10*, Nd.15	1636	0x664	RxPDO10*, Nd.36	1657	0x679	RxPDO10*, Nd.57
1616	0x650	RxPDO10*, Nd.16	1637	0x665	RxPDO10*, Nd.37	1658	0x67A	RxPDO10*, Nd.58
1617	0x651	RxPDO10*, Nd.17	1638	0x666	RxPDO10*, Nd.38	1659	0x67B	RxPDO10*, Nd.59
1618	0x652	RxPDO10*, Nd.18	1639	0x667	RxPDO10*, Nd.39	1660	0x67C	RxPDO10*, Nd.60
1619	0x653	RxPDO10*, Nd.19	1640	0x668	RxPDO10*, Nd.40	1661	0x67D	RxPDO10*, Nd.61
1620	0x654	RxPDO10*, Nd.20	1641	0x669	RxPDO10*, Nd.41	1662	0x67E	RxPDO10*, Nd.62
1621	0x655	RxPDO10*, Nd.21	1642	0x66A	RxPDO10*, Nd.42	1663	0x67F	RxPDO10*, Nd.63

dec	hex	Telegram type	dec	hex	Telegram type	dec	hex	Telegram type
1729	0x6C1	TxPDO11*, Nd.1	1750	0x6D6	TxPDO11*, Nd.22	1771	0x6EB	TxPDO11*, Nd.43
1730	0x6C2	TxPDO11*, Nd.2	1751	0x6D7	TxPDO11*, Nd.23	1772	0x6EC	TxPDO11*, Nd.44
1731	0x6C3	TxPDO11*, Nd.3	1752	0x6D8	TxPDO11*, Nd.24	1773	0x6ED	TxPDO11*, Nd.45
1732	0x6C4	TxPDO11*, Nd.4	1753	0x6D9	TxPDO11*, Nd.25	1774	0x6EE	TxPDO11*, Nd.46
1733	0x6C5	TxPDO11*, Nd.5	1754	0x6DA	TxPDO11*, Nd.26	1775	0x6EF	TxPDO11*, Nd.47
1734	0x6C6	TxPDO11*, Nd.6	1755	0x6DB	TxPDO11*, Nd.27	1776	0x6F0	TxPDO11*, Nd.48
1735	0x6C7	TxPDO11*, Nd.7	1756	0x6DC	TxPDO11*, Nd.28	1777	0x6F1	TxPDO11*, Nd.49
1736	0x6C8	TxPDO11*, Nd.8	1757	0x6DD	TxPDO11*, Nd.29	1778	0x6F2	TxPDO11*, Nd.50
1737	0x6C9	TxPDO11*, Nd.9	1758	0x6DE	TxPDO11*, Nd.30	1779	0x6F3	TxPDO11*, Nd.51
1738	0x6CA	TxPDO11*, Nd.10	1759	0x6DF	TxPDO11*, Nd.31	1780	0x6F4	TxPDO11*, Nd.52
1739	0x6CB	TxPDO11*, Nd.11	1760	0x6E0	TxPDO11*, Nd.32	1781	0x6F5	TxPDO11*, Nd.53
1740	0x6CC	TxPDO11*, Nd.12	1761	0x6E1	TxPDO11*, Nd.33	1782	0x6F6	TxPDO11*, Nd.54
1741	0x6CD	TxPDO11*, Nd.13	1762	0x6E2	TxPDO11*, Nd.34	1783	0x6F7	TxPDO11*, Nd.55
1742	0x6CE	TxPDO11*, Nd.14	1763	0x6E3	TxPDO11*, Nd.35	1784	0x6F8	TxPDO11*, Nd.56
1743	0x6CF	TxPDO11*, Nd.15	1764	0x6E4	TxPDO11*, Nd.36	1785	0x6F9	TxPDO11*, Nd.57
1744	0x6D0	TxPDO11*, Nd.16	1765	0x6E5	TxPDO11*, Nd.37	1786	0x6FA	TxPDO11*, Nd.58
1745	0x6D1	TxPDO11*, Nd.17	1766	0x6E6	TxPDO11*, Nd.38	1787	0x6FB	TxPDO11*, Nd.59
1746	0x6D2	TxPDO11*, Nd.18	1767	0x6E7	TxPDO11*, Nd.39	1788	0x6FC	TxPDO11*, Nd.60
1747	0x6D3	TxPDO11*, Nd.19	1768	0x6E8	TxPDO11*, Nd.40	1789	0x6FD	TxPDO11*, Nd.61
1748	0x6D4	TxPDO11*, Nd.20	1769	0x6E9	TxPDO11*, Nd.41	1790	0x6FE	TxPDO11*, Nd.62
1749	0x6D5	TxPDO11*, Nd.21	1770	0x6EA	TxPDO11*, Nd.42	1791	0x6FF	TxPDO11*, Nd.63

dec	hex	Telegram type	dec	hex	Telegram type	dec	hex	Telegram type
1857	0x741	RxPDO11*, Nd.1	1878	0x756	RxPDO11*, Nd.22	1899	0x76B	RxPDO11*, Nd.43
1858	0x742	RxPDO11*, Nd.2	1879	0x757	RxPDO11*, Nd.23	1900	0x76C	RxPDO11*, Nd.44
1859	0x743	RxPDO11*, Nd.3	1880	0x758	RxPDO11*, Nd.24	1901	0x76D	RxPDO11*, Nd.45
1860	0x744	RxPDO11*, Nd.4	1881	0x759	RxPDO11*, Nd.25	1902	0x76E	RxPDO11*, Nd.46
1861	0x745	RxPDO11*, Nd.5	1882	0x75A	RxPDO11*, Nd.26	1903	0x76F	RxPDO11*, Nd.47
1862	0x746	RxPDO11*, Nd.6	1883	0x75B	RxPDO11*, Nd.27	1904	0x770	RxPDO11*, Nd.48
1863	0x747	RxPDO11*, Nd.7	1884	0x75C	RxPDO11*, Nd.28	1905	0x771	RxPDO11*, Nd.49
1864	0x748	RxPDO11*, Nd.8	1885	0x75D	RxPDO11*, Nd.29	1906	0x772	RxPDO11*, Nd.50
1865	0x749	RxPDO11*, Nd.9	1886	0x75E	RxPDO11*, Nd.30	1907	0x773	RxPDO11*, Nd.51
1866	0x74A	RxPDO11*, Nd.10	1887	0x75F	RxPDO11*, Nd.31	1908	0x774	RxPDO11*, Nd.52
1867	0x74B	RxPDO11*, Nd.11	1888	0x760	RxPDO11*, Nd.32	1909	0x775	RxPDO11*, Nd.53
1868	0x74C	RxPDO11*, Nd.12	1889	0x761	RxPDO11*, Nd.33	1910	0x776	RxPDO11*, Nd.54
1869	0x74D	RxPDO11*, Nd.13	1890	0x762	RxPDO11*, Nd.34	1911	0x777	RxPDO11*, Nd.55
1870	0x74E	RxPDO11*, Nd.14	1891	0x763	RxPDO11*, Nd.35	1912	0x778	RxPDO11*, Nd.56
1871	0x74F	RxPDO11*, Nd.15	1892	0x764	RxPDO11*, Nd.36	1913	0x779	RxPDO11*, Nd.57
1872	0x750	RxPDO11*, Nd.16	1893	0x765	RxPDO11*, Nd.37	1914	0x77A	RxPDO11*, Nd.58
1873	0x751	RxPDO11*, Nd.17	1894	0x766	RxPDO11*, Nd.38	1915	0x77B	RxPDO11*, Nd.59
1874	0x752	RxPDO11*, Nd.18	1895	0x767	RxPDO11*, Nd.39	1916	0x77C	RxPDO11*, Nd.60
1875	0x753	RxPDO11*, Nd.19	1896	0x768	RxPDO11*, Nd.40	1917	0x77D	RxPDO11*, Nd.61
1876	0x754	RxPDO11*, Nd.20	1897	0x769	RxPDO11*, Nd.41	1918	0x77E	RxPDO11*, Nd.62
1877	0x755	RxPDO11*, Nd.21	1898	0x76A	RxPDO11*, Nd.42	1919	0x77F	RxPDO11*, Nd.63

dec	hex	Telegram type	dec	hex	Telegram type	dec	hex	Telegram type
1409	0x581	SDO Tx Nd.1	1430	0x596	SDO Tx Nd.22	1451	0x5AB	SDO Tx Nd.43
1410	0x582	SDO Tx Nd.2	1431	0x597	SDO Tx Nd.23	1452	0x5AC	SDO Tx Nd.44
1411	0x583	SDO Tx Nd.3	1432	0x598	SDO Tx Nd.24	1453	0x5AD	SDO Tx Nd.45
1412	0x584	SDO Tx Nd.4	1433	0x599	SDO Tx Nd.25	1454	0x5AE	SDO Tx Nd.46
1413	0x585	SDO Tx Nd.5	1434	0x59A	SDO Tx Nd.26	1455	0x5AF	SDO Tx Nd.47
1414	0x586	SDO Tx Nd.6	1435	0x59B	SDO Tx Nd.27	1456	0x5B0	SDO Tx Nd.48
1415	0x587	SDO Tx Nd.7	1436	0x59C	SDO Tx Nd.28	1457	0x5B1	SDO Tx Nd.49
1416	0x588	SDO Tx Nd.8	1437	0x59D	SDO Tx Nd.29	1458	0x5B2	SDO Tx Nd.50
1417	0x589	SDO Tx Nd.9	1438	0x59E	SDO Tx Nd.30	1459	0x5B3	SDO Tx Nd.51
1418	0x58A	SDO Tx Nd.10	1439	0x59F	SDO Tx Nd.31	1460	0x5B4	SDO Tx Nd.52
1419	0x58B	SDO Tx Nd.11	1440	0x5A0	SDO Tx Nd.32	1461	0x5B5	SDO Tx Nd.53
1420	0x58C	SDO Tx Nd.12	1441	0x5A1	SDO Tx Nd.33	1462	0x5B6	SDO Tx Nd.54
1421	0x58D	SDO Tx Nd.13	1442	0x5A2	SDO Tx Nd.34	1463	0x5B7	SDO Tx Nd.55
1422	0x58E	SDO Tx Nd.14	1443	0x5A3	SDO Tx Nd.35	1464	0x5B8	SDO Tx Nd.56
1423	0x58F	SDO Tx Nd.15	1444	0x5A4	SDO Tx Nd.36	1465	0x5B9	SDO Tx Nd.57
1424	0x590	SDO Tx Nd.16	1445	0x5A5	SDO Tx Nd.37	1466	0x5BA	SDO Tx Nd.58
1425	0x591	SDO Tx Nd.17	1446	0x5A6	SDO Tx Nd.38	1467	0x5BB	SDO Tx Nd.59
1426	0x592	SDO Tx Nd.18	1447	0x5A7	SDO Tx Nd.39	1468	0x5BC	SDO Tx Nd.60
1427	0x593	SDO Tx Nd.19	1448	0x5A8	SDO Tx Nd.40	1469	0x5BD	SDO Tx Nd.61
1428	0x594	SDO Tx Nd.20	1449	0x5A9	SDO Tx Nd.41	1470	0x5BE	SDO Tx Nd.62
1429	0x595	SDO Tx Nd.21	1450	0x5AA	SDO Tx Nd.42	1471	0x5BF	SDO Tx Nd.63

dec	hex	Telegram type	dec	hex	Telegram type	dec	hex	Telegram type
1537	0x601	SDO Rx Nd.1	1558	0x616	SDO Rx Nd.22	1579	0x62B	SDO Rx Nd.43
1538	0x602	SDO Rx Nd.2	1559	0x617	SDO Rx Nd.23	1580	0x62C	SDO Rx Nd.44
1539	0x603	SDO Rx Nd.3	1560	0x618	SDO Rx Nd.24	1581	0x62D	SDO Rx Nd.45
1540	0x604	SDO Rx Nd.4	1561	0x619	SDO Rx Nd.25	1582	0x62E	SDO Rx Nd.46
1541	0x605	SDO Rx Nd.5	1562	0x61A	SDO Rx Nd.26	1583	0x62F	SDO Rx Nd.47
1542	0x606	SDO Rx Nd.6	1563	0x61B	SDO Rx Nd.27	1584	0x630	SDO Rx Nd.48
1543	0x607	SDO Rx Nd.7	1564	0x61C	SDO Rx Nd.28	1585	0x631	SDO Rx Nd.49
1544	0x608	SDO Rx Nd.8	1565	0x61D	SDO Rx Nd.29	1586	0x632	SDO Rx Nd.50
1545	0x609	SDO Rx Nd.9	1566	0x61E	SDO Rx Nd.30	1587	0x633	SDO Rx Nd.51
1546	0x60A	SDO Rx Nd.10	1567	0x61F	SDO Rx Nd.31	1588	0x634	SDO Rx Nd.52
1547	0x60B	SDO Rx Nd.11	1568	0x620	SDO Rx Nd.32	1589	0x635	SDO Rx Nd.53
1548	0x60C	SDO Rx Nd.12	1569	0x621	SDO Rx Nd.33	1590	0x636	SDO Rx Nd.54
1549	0x60D	SDO Rx Nd.13	1570	0x622	SDO Rx Nd.34	1591	0x637	SDO Rx Nd.55
1550	0x60E	SDO Rx Nd.14	1571	0x623	SDO Rx Nd.35	1592	0x638	SDO Rx Nd.56
1551	0x60F	SDO Rx Nd.15	1572	0x624	SDO Rx Nd.36	1593	0x639	SDO Rx Nd.57
1552	0x610	SDO Rx Nd.16	1573	0x625	SDO Rx Nd.37	1594	0x63A	SDO Rx Nd.58
1553	0x611	SDO Rx Nd.17	1574	0x626	SDO Rx Nd.38	1595	0x63B	SDO Rx Nd.59
1554	0x612	SDO Rx Nd.18	1575	0x627	SDO Rx Nd.39	1596	0x63C	SDO Rx Nd.60
1555	0x613	SDO Rx Nd.19	1576	0x628	SDO Rx Nd.40	1597	0x63D	SDO Rx Nd.61
1556	0x614	SDO Rx Nd.20	1577	0x629	SDO Rx Nd.41	1598	0x63E	SDO Rx Nd.62
1557	0x615	SDO Rx Nd.21	1578	0x62A	SDO Rx Nd.42	1599	0x63F	SDO Rx Nd.63

dec	hex	Telegram type	dec	hex	Telegram type	dec	hex	Telegram type
1793	0x701	Guarding Nd.1	1814	0x716	Guarding Nd.22	1835	0x72B	Guarding Nd.43
1794	0x702	Guarding Nd.2	1815	0x717	Guarding Nd.23	1836	0x72C	Guarding Nd.44
1795	0x703	Guarding Nd.3	1816	0x718	Guarding Nd.24	1837	0x72D	Guarding Nd.45
1796	0x704	Guarding Nd.4	1817	0x719	Guarding Nd.25	1838	0x72E	Guarding Nd.46
1797	0x705	Guarding Nd.5	1818	0x71A	Guarding Nd.26	1839	0x72F	Guarding Nd.47
1798	0x706	Guarding Nd.6	1819	0x71B	Guarding Nd.27	1840	0x730	Guarding Nd.48
1799	0x707	Guarding Nd.7	1820	0x71C	Guarding Nd.28	1841	0x731	Guarding Nd.49
1800	0x708	Guarding Nd.8	1821	0x71D	Guarding Nd.29	1842	0x732	Guarding Nd.50
1801	0x709	Guarding Nd.9	1822	0x71E	Guarding Nd.30	1843	0x733	Guarding Nd.51
1802	0x70A	Guarding Nd.10	1823	0x71F	Guarding Nd.31	1844	0x734	Guarding Nd.52
1803	0x70B	Guarding Nd.11	1824	0x720	Guarding Nd.32	1845	0x735	Guarding Nd.53
1804	0x70C	Guarding Nd.12	1825	0x721	Guarding Nd.33	1846	0x736	Guarding Nd.54
1805	0x70D	Guarding Nd.13	1826	0x722	Guarding Nd.34	1847	0x737	Guarding Nd.55
1806	0x70E	Guarding Nd.14	1827	0x723	Guarding Nd.35	1848	0x738	Guarding Nd.56
1807	0x70F	Guarding Nd.15	1828	0x724	Guarding Nd.36	1849	0x739	Guarding Nd.57
1808	0x710	Guarding Nd.16	1829	0x725	Guarding Nd.37	1850	0x73A	Guarding Nd.58
1809	0x711	Guarding Nd.17	1830	0x726	Guarding Nd.38	1851	0x73B	Guarding Nd.59
1810	0x712	Guarding Nd.18	1831	0x727	Guarding Nd.39	1852	0x73C	Guarding Nd.60
1811	0x713	Guarding Nd.19	1832	0x728	Guarding Nd.40	1853	0x73D	Guarding Nd.61
1812	0x714	Guarding Nd.20	1833	0x729	Guarding Nd.41	1854	0x73E	Guarding Nd.62
1813	0x715	Guarding Nd.21	1834	0x72A	Guarding Nd.42	1855	0x73F	Guarding Nd.63

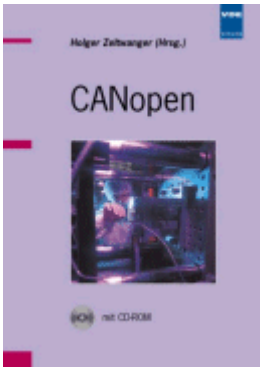
7.5 Abbreviations

Abbreviation	Description
CAN	Controller Area Network. Serial bus system standardized in ISO 11898 that is used as the basic technology for CANopen
CiA	CAN in Automation e.V.. An international association of manufacturers and users based in Erlangen, Germany.
CoB	Communication Object. A CAN telegram with up to 8 data bytes.
CoB-ID	Communication Object Identifier. Telegram address (not to be confused with the node address). CANopen uses the 11-bit identifier according to CAN 2.0A.
CoE	CANopen over EtherCAT
ESC	EtherCAT Slave Controller
FBM	Fieldbus master
MC	Motion Control
NMT	Network Management. One of the service primitives of the CANopen specification. Network management is used to initialize the network and to monitor nodes.
OP	OPERATIONAL
PDO	Process Data Object. A CAN telegram for the transfer of process data (e.g. I/O data).
PREOP	PRE-OPERATIONAL
RxPDO	Receive PDO. PDOs are always identified from the point of view of the device under consideration. Thus a TxPDO with input data from an I/O module becomes an RxPDO from the controller's point of view.
SAFEOP	SAFE OPERATIONAL
SDO	Service Data Object. A CAN telegram with a protocol for communication with data in the object directory (typically parameter data).
SI	Subindex
SM	SyncManager
SoE	Servo Profile over EtherCAT
TxPDO	Transmit PDO (named from the point of view of the CAN node).

7.6 Bibliography

German books

- Holger Zeltwander (Pub.):
  CANopen,
  VDE Verlag, 2001. 197 pages,
  ISBN 3-800-72448-0



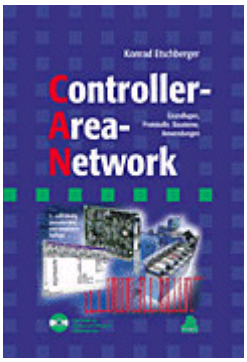
- Konrad Etschberger:
  Controller Area Network, Grundlagen, Protokolle, Bausteine, Anwendungen. (Principles, protocols, components, applications.)
  Hanser Verlag, 2000. 431 pages.
  ISBN 3-446-19431-2

General fieldbus technology

- Gerhard Gruhler (Pub.):
  Feldbusse und Geräte-Kommunikationssysteme, Praktisches Know-How mit Vergleichsmöglichkeiten. (Fieldbus and Device Communication Systems, Practical Know-how with Comparative Resources)
  Franzis Verlag, 2001. 244 pages.
  ISBN 3-7723-5745-8

English books

- Konrad Etschberger:
  Controller Area Network,
  Ixxat Press, 2001. 440 pages.
  ISBN 3-00-007376-0
- M. Farsi, M. Barbosa:
  CANopen Implementation,
  RSP 2000. 210 pages.
  ISBN 0-86380-247-8



Standards

- ISO 11898:
  Road Vehicles - Interchange of digital information - Controller Area Network (CAN) for high speed communication.
- CiA DS 301:
  CANopen Application Layer and Communication Profile.
  Available from the CAN in Automation Association.
- CiA DS 401:
  CANopen Device Profile for Generic I/O Modules.
  Available from the CAN in Automation Association.

7.7 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 5246 963 0
Fax:	+49 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 5246 963 157
Fax:	+49 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 5246 963 460
Fax:	+49 5246 963 479
e-mail:	service@beckhoff.com

Table of figures