EtherCAT and CANopen manual

Important!
Read thoroughly before use!
Retain for future reference!

Contact

Motor Power Company S.r.l.
Via Leonardo Da Vinci, 4
42024 Castelnovo Sotto - Reggio Emilia
Italy

Tel +39 0522 682710
Fax +39 0522 683552
Email: info@motorpowerco.it
https://www.motorpowerco.com

Manufacturer:
Metronix Meßgeräte und Elektronik GmbH,
Kocherstraße 3, 38120 Braunschweig, Germany

Revision Information

Manual title	EtherCAT and CANopen manual
Version	1.0
Year	2023

Table of Contents

1 About this manual

This manual describes how the servo drives of the BL 4000-C or BL 4000-M / BL 4000-D device series can be integrated into a CANopen or Ethercat network. The physical connection, activation of the fieldbus protocol, integration into the network and the parameters for adaptation to the respective application are described. It is intended for persons who are already familiar with the respective servo drive series and have read and understood the corresponding product manual.

The product manual contains instructions for the proper and professional transport, storage, assembly, installation, project planning and correct and safe operation of the servo drive.

The product manual contains safety instructions which must be strictly observed.

The product manuals are available for download on our homepage (https://www.motorpowerco.com).

1.1 Structure of the warning notes

Warning notes have the following structure:

- Signal word
- Type of hazard
- Measures to prevent the hazard

Signal words

⚠ DANGER

Indicates an imminent hazard.
If the situation is not avoided, extremely serious and possibly fatal injuries will result.

⚠ WARNING

Indicates a potentially hazardous situation.
If the situation is not avoided, extremely serious and possibly fatal injuries may result.

⚠ CAUTION

Indicates a potentially hazardous situation.
If the situation is not avoided, slight or minor injuries may result.

NOTICE

Warns against damage to property.

Warning signs as per ISO 7010

Warning sign	Explanation
	Warning against fatal electric voltage.

1.2 Notation in this manual

Structure of notes

The notes in this manual have the following structure:

- Signal word "NOTE"
- Introductory phrase
- Explanations and special tips

Operating elements, menus

Operating elements, menus and menu paths are written in red.

Example: Double-clicking the desired device or clicking the button Establish connection will establish an online connection.

CAN Objects, bit constants

Terms from the CANopen standards such as parameter names (CAN objects) are written in dark red. Bit constants are highlighted by a different font.

Example: If this bit is set, bit 4 of the statusword (`voltage_enabled`) is output according to DSP 402 v2.0.

States, commands

Servo drive states (see section 4 *Device Control* on page 102) are set in a different font and are capitalised. Commands are highlighted with a white box.

Example:

`NOT_READY_TO_SWITCH_ON`	The servo drive carries out a self-test.

4	Enable Operation	1	1	1	1	Motor control according to the current operating mode

2 Quick-start guide

This chapter describes how to connect the servo drives to a commercially available CANopen or Ethercat controller and put them into operation in order to obtain a quick setup for starting application development. Depending on which fieldbus interface is used, the respective other chapter can be skipped.

Section 3 *Parameterisation* on page 36 then describes all available parameters, which can usually be used equally under CANopen and EtherCAT, in order to adapt the servo drive to the respective application. This chapter is intended for users who already have an industrial controller.

2.1 CANopen

CANopen is a standard maintained by the association "CAN in Automation", which defines the use of CAN in automation technology independently of manufacturers. The CANopen interface in the BL 4000 is designed according to CiA 301 (transmission layer) and CiA 402 (drive controller profile).

2.1.1 Basics

The CANopen fieldbus protocol defines how data is exchanged via the CAN fieldbus in industrial automation.

In general, there are two types of messages (communication objects) that are exchanged between the master (e.g. CoDeSys controller) and the slave.

- **SDO (Service Data Objects)**
  This type of message is used for acyclic communication between master and slave, e.g. during the initialization phase of the application or in a very simple application where no cyclic data exchange is required.
- **PDO (Process Data Objects)**
  This type of message is exchanged cyclically/automatically between master and slave to transfer process data. Process data is all the data required by the master or slave to execute the application. In our example, this process data contains e.g. position setpoint/actual values, control and status words and other important information to be able to use the servo drive as a SoftMotion axis.

There are further message types, such as Emergency Messages, Heartbeat Messages or Node Guarding Messages, which are also exchanged between master and slave, but only in case of a special event or in special applications. For example, an Emergency Message is sent from the slave to the master when a serious error has occurred in the servo drive. A detailed description of these message types can be found in section 6 *Detailed description of the CANopen protocol* on page 160.

2.1.2 Wiring and pin assignment

The CAN interface is integrated in the BL 4000-C servo drives and therefore always available. For servo drives of the BL 4000-M / BL 4000-D series, the CAN interface is only available with the CAN field bus variant. More detailed information on this can be found in the *Product Description* section of the Product manual BL 4000-D and BL 4000-M.

INFORMATION **CAN bus wiring**

When wiring the servo drive via the CAN bus, it is essential that you observe the following information and notes in order to obtain a stable, trouble-free system.

If the cabling is not correct, faults can occur on the CAN bus during operation, which can cause the servo drive to switch off with a fault for safety reasons.

INFORMATION **120Ω terminating resistor**

No terminating resistor is integrated in the BL 4000-C, BL 4000-D and BL 4000-M servo drives.

BL 4000-C

The CAN bus connection is designed as a 9-pin DSUB connector (servo drive side) according to the standard.

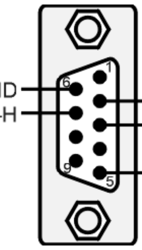


Figure 1: CAN connector

Pin		Name	Description
1			Not used
	6	CAN-GND	CAN-GND (directly coupled to GND in the BL 4000-C servo drive)
2		CAN-L	CAN low signal line
	7	CAN-H	CAN high signal line
3		CAN-GND	See pin no. 6
	8		Not used
4			Not used
	9		Not used
5		Shield	Connection for cable shield

BL 4000-D and BL 4000-M (Fieldbus variant CAN)

On these devices, the CAN bus connection is designed as an M8 connector according to IEC 61076-114 (4-pin, socket, D-coded). Note that although the fieldbus variant PROFINET/EtherCAT uses the identical connectors, it is not electrically compatible. The fieldbus variants must not be mixed up and must never be used simultaneously in the same network!

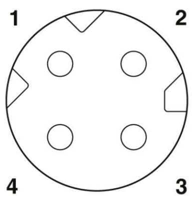


Figure 2: Pin assignment of the fieldbus connector

Pin assignment **CANopen**:

Pin	Name	Description	Colour
1	CAN-H	Differential Signal High	Yellow
2	CAN-GND	Reference potential	Orange
3	CAN-L	Differential Signal High	White
4	CAN-GND	Reference potential	Blue

For wiring, we recommend using the following pre-assembled cables or comparable products from other manufacturers:

Assembled network cable Phoenix Contact:

M8 plug to M8 plug: NBC-M8MSD/ 1,0-93C/M8MSD - 1423707
M8 plug to RJ45: NBC-M8MSD/ 1,0-93C/R4AC - 1423711
M8 plug to free cable end: NBC-M8MSD/ 1,0-93C - 1423703

2.1.3 Wiring instructions

For trouble-free operation of the CAN bus communication, the following instructions must be observed

- Ideally, the individual nodes of the networks are always connected in a linear manner so that the CAN cable is looped through from servo drive to servo drive.
- A terminating resistor of 120 Ω, 5%, must be present on both ends of the CAN bus cable
- We advise against the use of intermediate plugs for cabling the CAN bus. However, if this is necessary, use metal connector housings for connecting the cable shield.

In order to keep interferences as low as possible ensure that

- the motor cables are not installed parallel to signal lines
- the motor cables comply with the specification
- the motor cables are properly shielded and earthed (grounded)

The cable used should be constructed as follows

- Shielded cables with exactly two twisted pairs must be used for cabling
- Use one twisted pair to connect CAN-H and CAN-L.
- The cores of the other pair are used jointly for CAN-GND.
- The shield of the cable is led to the CAN shield connections for all nodes

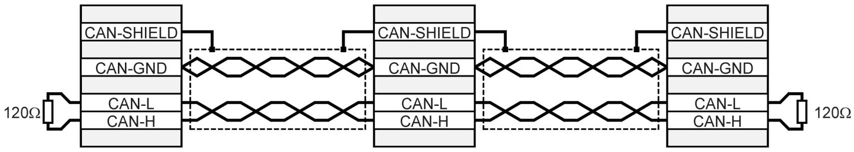


Figure 3: CAN bus cabling example

› Technical data CAN cable

The cable names that are stated refer to cables made by Lapp. However, it is also possible to use comparable cables from other manufacturers, for example Lütze or Helukabel.

Technical data of the CAN bus cable: 2 pairs of 2 twisted cores, d ≥ 0.22 mm^2, shielded, loop resistance < 0.2 Ω/m, characteristic impedance 100-120 Ω

LAPP KABEL UNITRONIC BUS CAN; 2 x 2 x 0.22; 7.6 mm, with CU shielding

For highly flexible applications:

LAPP KABEL UNITRONIC BUS CAN FD P; 2 x 2 x 0.25; 8.4 mm, with CU shielding

2.1.4 Status LEDs

› BL 4000-C

For easy indication of the CAN bus status, the servo drive is equipped with two fieldbus status LEDs:

The LEDs indicate the following states:

Name	Colour	Description
RUN/SF/MS	Green	This LED indicates ongoing communication between the master and the servo drive. It is triggered when a message is received from the master. If this LED is continuously OFF, there is no communication with the servo drive.
ERR/BF/NS	Red	This LED indicates the fieldbus error related to the CAN fieldbus. The LED flashes if a CAN-related fieldbus error is present and has not yet been acknowledged.

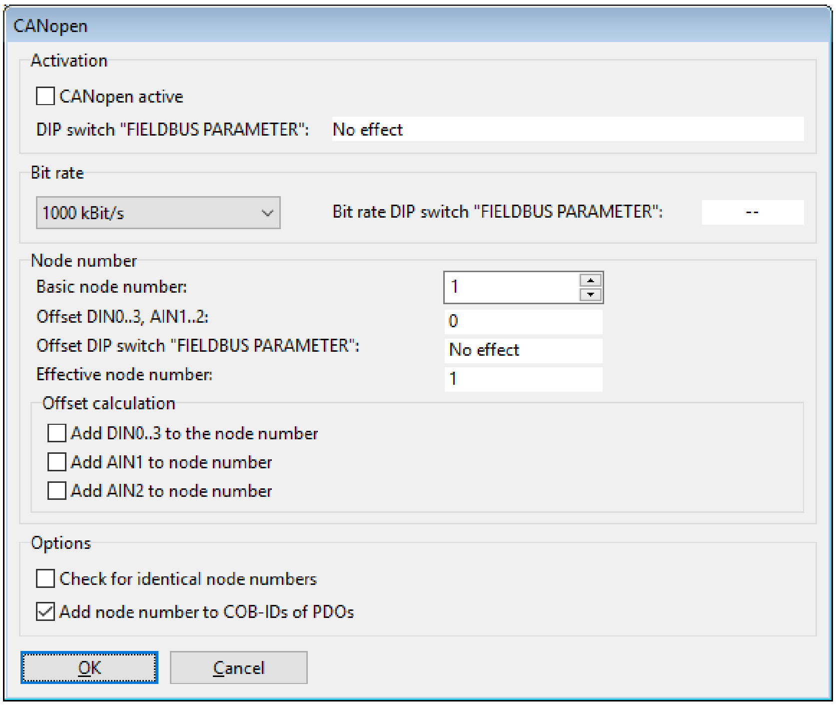
In normal operation the RUN LED is on, because communication with the servo drive is taking place and the ERR LED is off.

If the ERR LED is flashing, one of the following CAN fieldbus errors has occurred:

Group 12: CAN communication		
12-1	CAN: communication error, bus OFF	Check the wiring: compliance with the cable specification, cable break, maximum cable length exceeded, correct terminating resistors, cable shield earthed, all signals connected?
12-2	CAN: communication error (sending)	
12-3	CAN: communication error (receiving)	
12-4	CAN: Node Guarding	Failure of the PLC or the cycle time of the remote frames of the servo drive and PLC do not match.
12-5	CAN: RPDO too short	The number of bytes of a received RPDO is smaller than the number that is parameterised in the servo drive.
12-9	CAN: protocol error	Please contact the Technical Support team.

2.1.5 Activate CANopen

The CANopen fieldbus communication must be activated once via the CANopen window of the MPC ServoCommander (Parameter / Fieldbus / CANopen / Operation parameters). Depending on the devices series, not all options may be available, so that the appearance of the window may be different.



A total of 3 different parameters must be set:

Parameter	Description
Bit rate	This parameter determines the bit rate in kBit/s used on the CAN bus. It must match the bit rate in the master. Note that the maximum permissible cable length decreases at high bit rates.

Parameter	Description
Node number	For clear identification in the network, each participant must be assigned a node number that may only occur once in the network. The device is addressed via this node number. As an additional option it is possible to make the node number of the servo drive dependent on the external connection. The input combination of the digital inputs DIN0...DIN3 is added once to the basic node number after the reset.
Options	**Add node number to COB-IDs of the PDOs:** By setting this option, the COB-IDs of the PDOs do not have to be adapted manually to the node number (see section 6.3.2 *Objects for PDO configuration* on page 168).

Finally, the CANopen protocol can be activated. The above mentioned parameters can only be changed if the protocol is deactivated.

INFORMATION **Parametrisation of the CANopen functionality**

Please note that the parametrisation of the CANopen functionality is only retained after a reset if the parameter set of the servo drive has been saved.

INFORMATION **Identical node numbers**

It is not permitted to operate several servo drives on the CANopen fieldbus with the same node number. Therefore, make sure that each servo drive on the CANopen fieldbus has a unique node number before you activate communication.

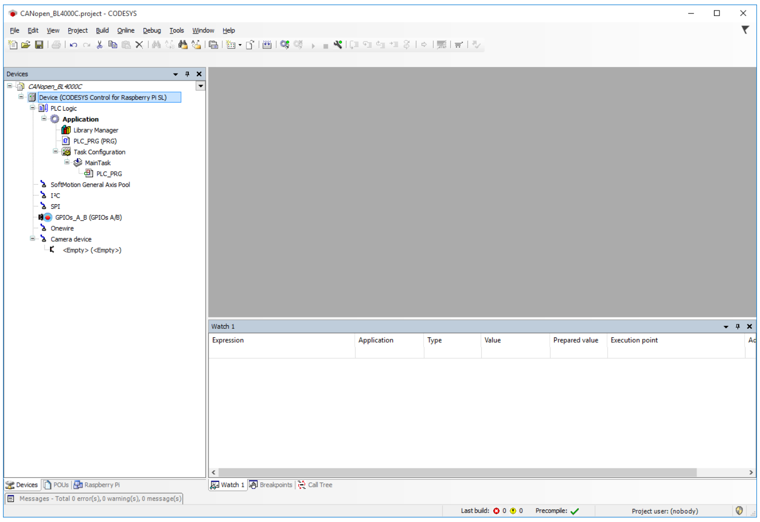
2.1.6 Integration of the servo drive in a master project

As an example, this chapter shows how to integrate the servo drive into a CoDeSys V3.5 project and operate them as SoftMotion drives.

As a prerequisite, you must download the CANopen EDS file (Electronic Data Sheet) for the corresponding servo drive from our website (https://www.motorpowerco.com). This file contains a complete description of the drive characteristics and the object dictionary and is used by CoDeSys (or another CANopen master) for the automatic configuration of the servo drive. The following example shows the installation of a BL 4104-C.

› Create a new project

Start CoDeSys, connect to your CANopen master and create an empty project.



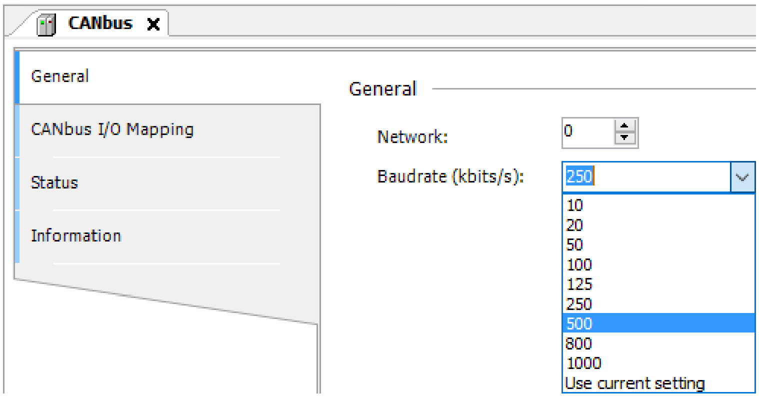
› Install the EDS file in the CoDeSys device directory

- Open the CoDeSys device directory.
  Path: Tools / Device Repository
- Click the Install button
- Select the downloaded EDS file from your location.
- Confirm by clicking the Open button

Now the CoDeSys software knows the servo drive and it can be used.

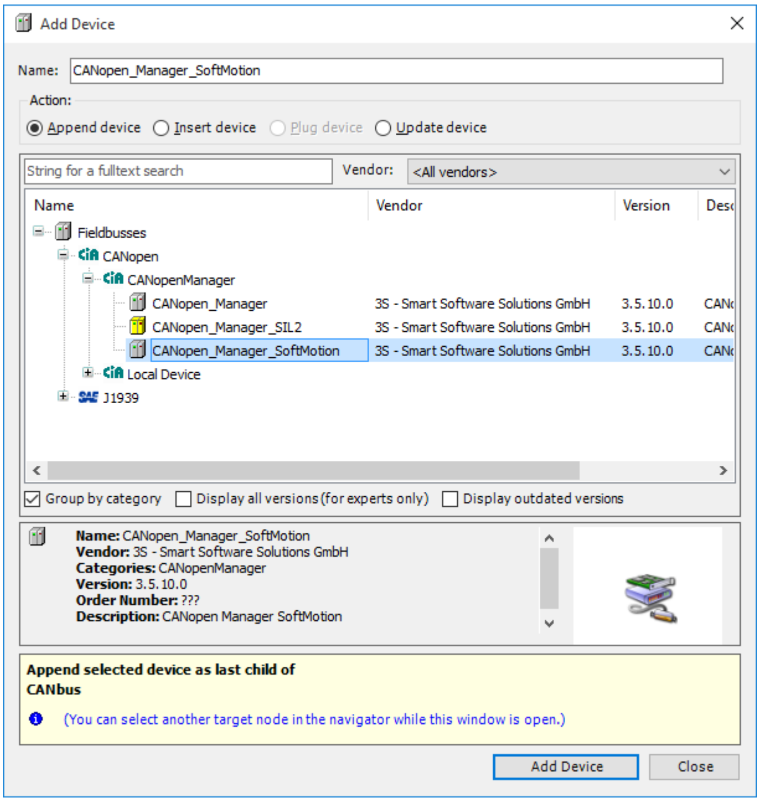
Add CANopen Master

Next, a CANopen master must be added. Therefore right click on the Master device and select Add Device.The CAN Master must be configured to the same bit rate as the one selected for the servo drive via the MPC ServoCommander.



To be able to connect the servo drive to the CAN master, an additional CANopen SoftMotion Manager must be added to the CAN master.

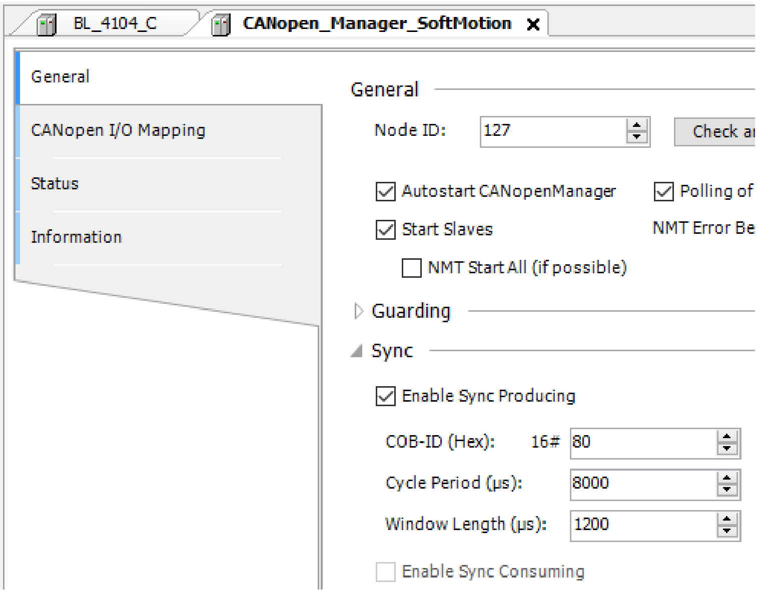
To do this, right click on the CAN master again and select Add Device.



Set cycle period

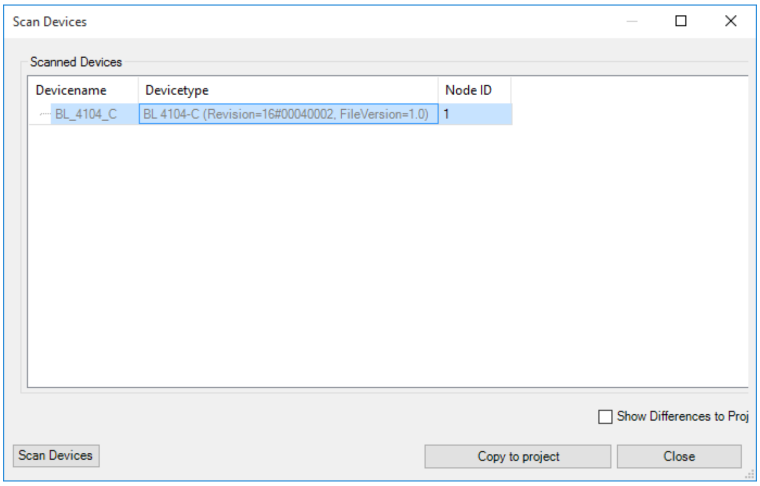
The SoftMotion Manager runs with a specific cycle time. As cyclic PDO data exchange is used in our application, the master synchronises the servo drive to this cycle time. To do this, the cycle time of the master (Cycle Period) must match the cycle time configured in the servo drive.

In the MPC ServoCommander you will find the dialog for configuring the cycle time in the menu Parameters\Controller parameters\Cycle times. For more information on setting the cycle times, refer to the section *Control circuit cycle times* in the product manual BL 4000.



Adding devices to the project

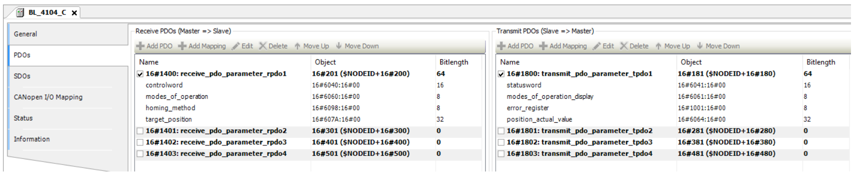
Finally, the generation of synchronisation telegrams must be activated in the SoftMotion Manager (Enable Sync Producing). Log on to the master by clicking on the Online Config Mode button. Search for servo drives on the CANopen field bus by right-clicking on the CANopen SoftMotion Manager and selecting Scan devices.



All servo drives connected to the fieldbus are detected and can be added to the project by clicking the Copy to project button. Afterwards the selected servo drives are displayed as devices connected to SoftMotion Manager.

Set PDO configuration

After the servo drive has been found, the cyclic data to be exchanged between servo drive and master must be specified. This is called PDO configuration and can be found on the tab with the corresponding servo drive name (in this case BL 4104-C).



The standard PDO mapping only uses the PDOs 1400_h (TPDO0-Master►Slave) and 1800_h (RPDO0-Master◄Slave).

These PDOs contain the following parameters for operating the servo drive as a SoftMotion axis:

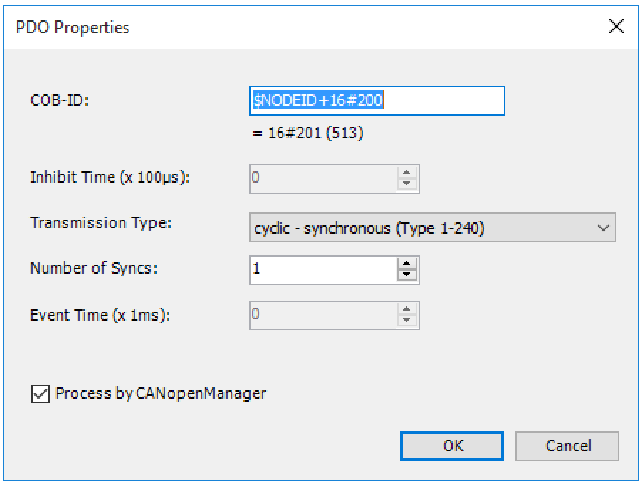
TPDO 0

Name	ID	Description	See
controlword	6040_h	Control word for activating / deactivating the servo drive	page 102
modes_of_operation	6060_h	Configuration of the operating mode of the drive	page 122
homing_method	6098_h	Configuring the homing method to be used	page 124
target_position	$607A_h$	Position setpoints	page 147

RPDO 0

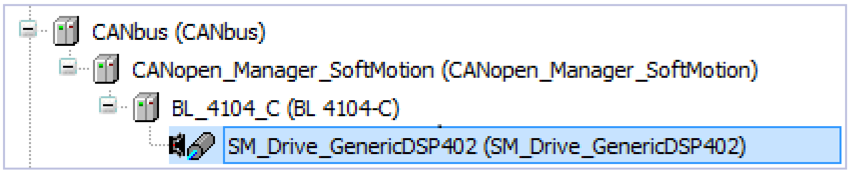
Name	ID	Description	See
statusword	6041_h	Current status of the drive	page 102
modes_of_operation_ display	6061_h	Current operation mode of the drive	page 122
error_register	1001_h	Current error code of the drive	page 174
position_actual_value	6064_h	Actual position value	page 66

Both PDOs must be set to "Cyclic transmission at 1 Sync". This is done by selecting the corresponding PDO and clicking the Edit button.



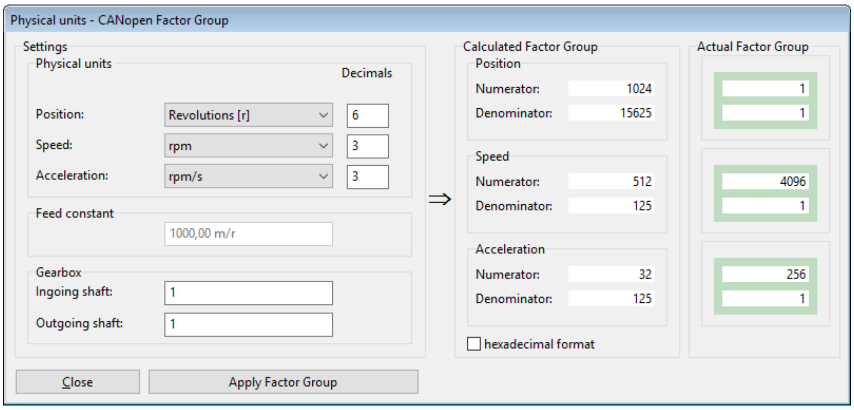
Once the PDO configuration is complete, a SoftMotion axis can be added to the servo drive.

To do this, select the servo drive in SoftMotion Manager. A context menu opens by right-clicking on the list entry (BL _4104_C). Click on the menu item Add SoftMotion CiA402 Axis.



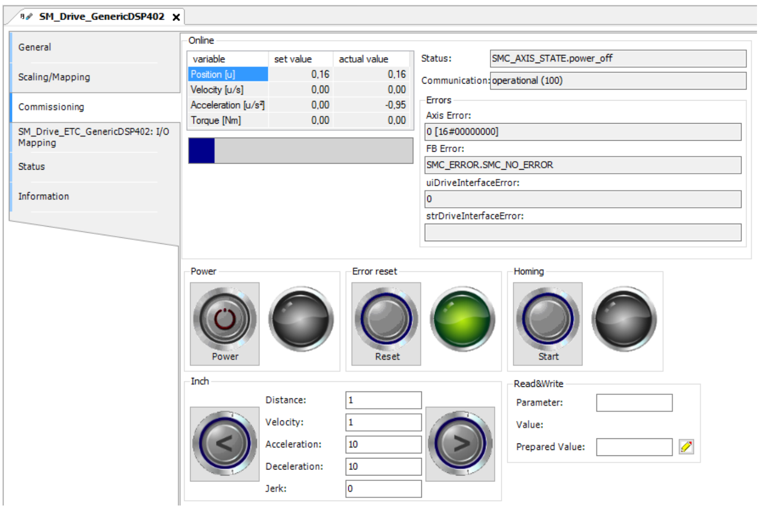
› Adjust conversion factors

To ensure that the units of the cyclically exchanged values (e.g. for position and speed) match between master and servo drive, the scaling in the servo drive must be adapted to the scaling in the master. Therefore the following scaling should be set in the MPC ServoCommander under Parameters/Field bus/CANopen/Display units:



With this scaling a maximum of +-32768 revolutions with 16 bit resolution can be displayed on the bus. If this is not sufficient, the scaling of the setpoints transmitted on the bus can be adapted. This is described in section 3.3 *Factor Group* on page 43.

If the scaling is configured correctly, the actual position values should now be displayed in the Commissioning window of the CoDeSys SoftMotion axis:



The axis can now be moved from the Commissioning tab for testing. In addition, the axis is now ready for implementation in the PLC project. A detailed description of all parameters of the servo drive and the implemented operating modes can be found in section 3 *Parameterisation* on page 36.

2.2 EtherCAT

EtherCAT is a real-time Ethernet developed by Beckhoff Automation. The *CAN application protocol over EtherCAT* (*CoE*) has been defined to enable an easy changeover from CAN to EtherCAT. This allows the CiA 402 drive controller profile to be used via EtherCAT.

2.2.1 Basics

CoE is based on the CANopen field bus protocol and therefore uses the same object dictionary and the same message types:

- **SDO (Service Data Objects)**This type of message is used for acyclic communication between master and slave, e.g. during the initialization phase of the application or in a very simple application where no cyclic data exchange is required.
- **PDO (Process Data Objects)**
  This type of message is exchanged cyclically/automatically between master and slave to exchange process data. Process data is all the data required by the master or slave to execute the application. In our example, this process data contains e.g. position setpoint/actual values, control and status words and other important information to be able to use the servo drive as a SoftMotion axis.

The message type Emergency Message is also available. This message is sent from the slave to the master if a serious error has occurred in the servo drive.

Other message types, such as Sync messages, are not supported by EtherCAT CoE, because there are other mechanisms to synchronise several slaves on the fieldbus to a common clock. The most important one is Distributed Clocks (DC), which are fully supported by the BL 4000 devices series.

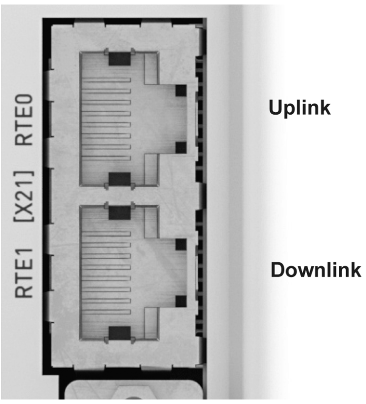
Synchronisation is important for motion applications in which several drives execute interpolated movements.

2.2.2 Wiring and pin assignment

In the BL 4000-C servo drives the EtherCAT interface is already integrated in the servo drives. For servo drives of the BL 4000-M / BL 4000-D series, the EtherCAT interface is only available with the PROFINET/EtherCAT field bus variant. For more detailed information, see the *Product Description* section of the Product manual BL 4000-D and BL 4000-M.

› BL 4000-C

According to the EtherCAT specification, two RJ45 connectors are available as RTE0 and RTE1 [X21]. One for uplink (connection from the previous drive) and one as downlink (connection to the next servo drive in the line).



The two connections RTE0 and RTE1 are RJ45 sockets, Cat. 6

Pin	Designation	Description
1	RX-	Receiver signal -
2	RX+	Receiver signal +
3	TX-	Transmitter signal -
4	-	-
5	-	-
6	TX+	Transmitter signal +
7	-	-
8	-	-

› BL 4000-D and BL 4000-M (Fieldbus variant PROFINET/EtherCAT)

On these devices, the EtherCAT connection is designed as an M8 connector according to IEC 61076-114 (4-pin, socket, D-coded). Note that although the fieldbus variant CAN uses the identical connectors, it is not electrically compatible. The fieldbus variants must not be mixed up and must never be used simultaneously in the same network!

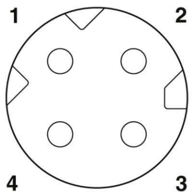


Figure 4: Pin assignment of the fieldbus connector

Pin assignment **EtherCAT/PROFINET**:

Pin	Name	Description	Colour
1	TD+	Transmission signal +	Yellow
2	RD+	Reception signal +	White
3	TD-	Transmission signal -	Orange
4	RD-	Reception signal -	Blue

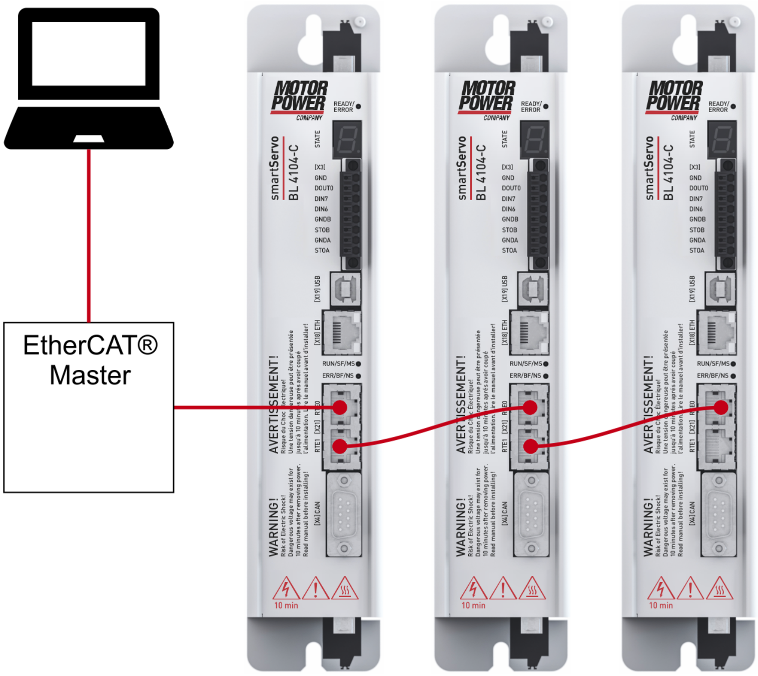
We recommend using the following pre-assembled cables or comparable products from other manufacturers.

Assembled network cable Phoenix Contact:

M8 plug to M8 plug: NBC-M8MSD/ 1,0-93C/M8MSD - 1423707
M8 plug to RJ45: NBC-M8MSD/ 1,0-93C/R4AC - 1423711
M8 plug to free cable end: NBC-M8MSD/ 1,0-93C - 1423703

2.2.3 Wiring instructions

For cabling, shielded twisted pair Ethernet cables that comply with STP, Cat.5 are used for the EtherCAT bus. All nodes of a network are connected in a linear manner.



2.2.4 Status LEDs (BL 4000-C)

For easy indication of the EtherCAT bus status, the servo drive series BL 4000-C is equipped with two fieldbus status LEDs. The behaviour of the LEDs is predefined by the EtherCAT User Group (ETG).

The green RUN LED indicates the current EtherCAT® CoE state:

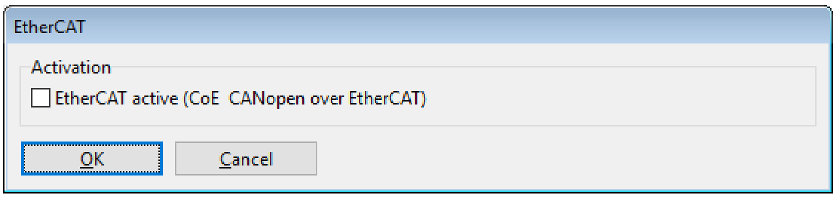
Flashing code	Status of the State Machine
LED is off	No communication yet.
LED flashes	**Pre-Operational (PreOp)** The master sets up the slave for cyclic communication. Only asynchronous communication via SDOs is active.
LED flashes once	**Safe Operation (SafeOp)** Cyclic communication via PDOs is running. The slave ignores the setpoint data, but sends actual values to the master.
LED is on	**Operational (OP)** The slave accepts setpoints from the master and follows them.

The red ERR-LED indicates possible fieldbus errors:

Flashing code	Status of the State machine
LED is off	No error
LED flashes twice	**Cyclic process data watchdog error** The fieldbus communication is interrupted. The slave has not received setpoints from the master.

2.2.5 Activate EtherCAT

The EtherCAT fieldbus communication must be activated once via the EtherCAT window of the MPC ServoCommander (Parameters / Field bus / EtherCAT / Operating parameters).



INFORMATION **Servo drive blocks communication to succeeding slaves**

Note that a servo drive with a deactivated Ethercat interface blocks communication to all following slaves on the fieldbus. Therefore a deactivated servo drive should be removed from the network.

2.2.6 Integration of the servo drive in a master project

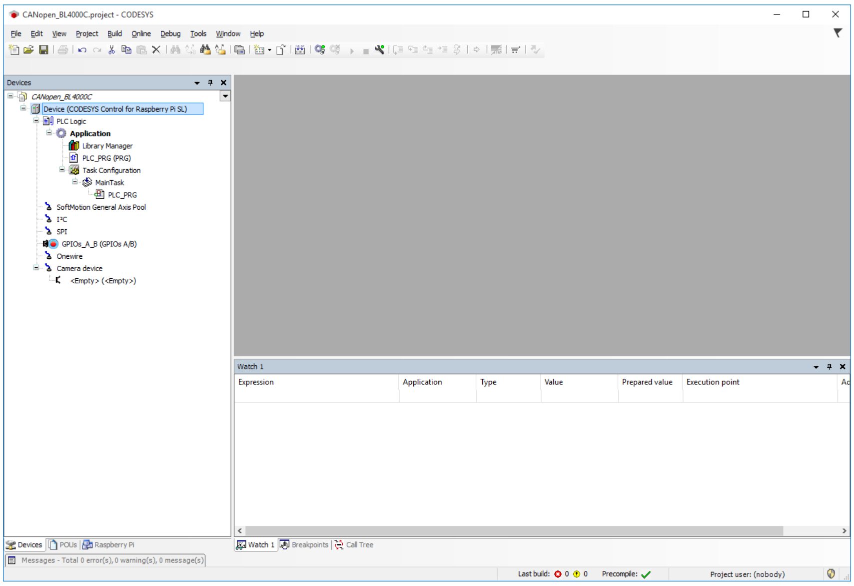
As an example, the servo drive BL 4104-C is to be inserted as a SoftMotion axis in a PLC project based on CoDeSys V3.5 and Beckhoff TwinCAT. The integration of other servo drives is done in the same way.

› Integration into the CoDeSys V3.5 project

As a prerequisite you must download the EtherCAT ESI file for the corresponding servo drive from our website (https://www.motorpowerco.com). This file contains a complete description of the drive features and the object dictionary and is used by CoDeSys (or any other EtherCAT master) to automatically configure the servo drive.

In contrast to the CANopen EDS file, this file contains not only the object dictionary, but also the complete configuration of the servo drive, including the selection of cyclically exchanged setpoints and actual values via PDOs, the configuration of the fieldbus cycle time and all necessary initialization commands to be sent to the servo drive when the fieldbus is started up.

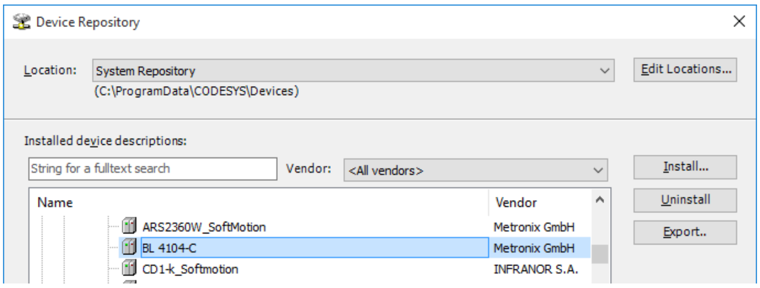
- After downloading the ESI XML file, connect the servo drive to the CoDeSys master via an Ethernet cable.
- Start CoDeSys, connect to your EtherCAT master and create an empty project.



› Install the ESI XML file in the CoDeSys device directory

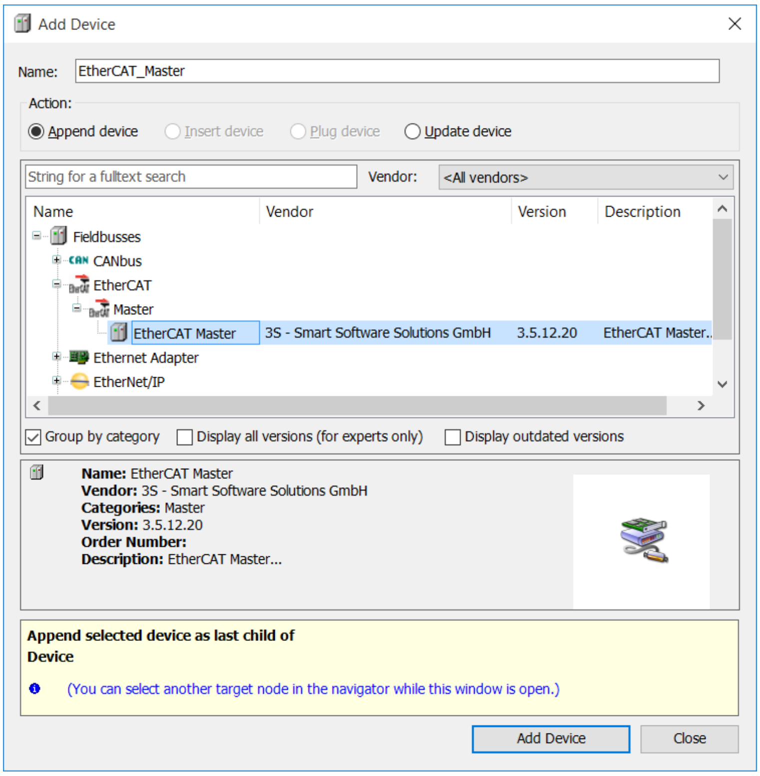
- Call up the CoDeSys device directory. **Path:** Tools / Device Repository
- Click the Install button.
- Select the downloaded EDS file from your location.
- Confirm by clicking the Open button.

Now the CoDeSys software knows the servo drive BL 4000-C and it can be used.



Add EtherCAT Master

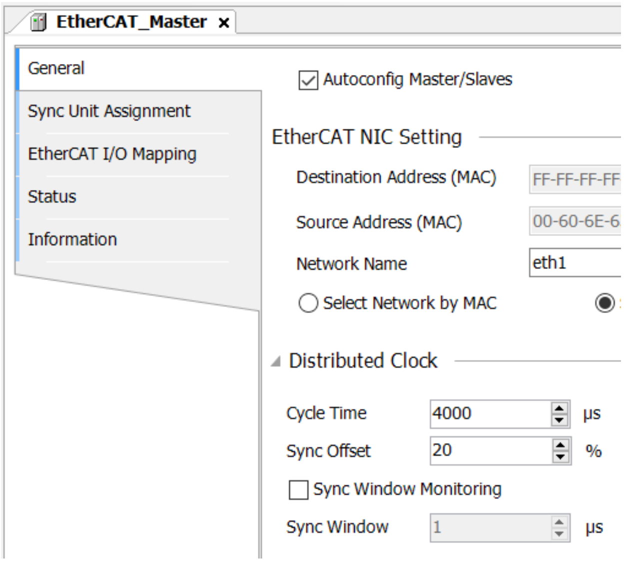
Next an EtherCAT Master must be added. To do this, right click on the Master device and select Add Device.



Set cycle time

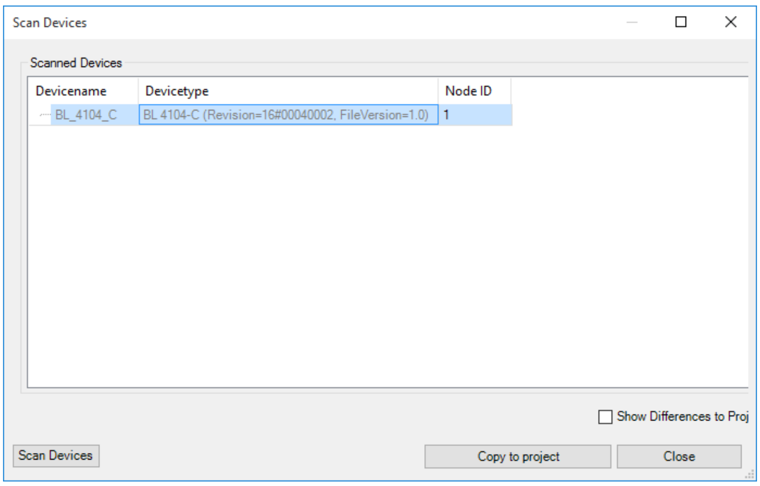
The EtherCAT master exchanges PDOs with the servo drive at a certain cycle time. For this purpose, the servo drive is synchronised by the master to this cycle time. The Cycle Time of the servo drive must therefore correspond to the cycle time configured in the EtherCAT master, and Distributed Clock (DC) must be activated in the master.

In the MPC ServoCommander you will find the dialog for configuring the cycle time in the menu Parameters\Controller parameters\Cycle times. For more information on setting the cycle times, refer to the section *Control circuit cycle times* in the product manual BL 4000.



Adding devices to the project

Finally, the generation of synchronisation telegrams must be activated in the SoftMotion Manager (Enable Sync Producing). Log on to the master by clicking on the Online Config Mode button. Search for servo drives on the CANopen field bus by right-clicking on the EtherCAT SoftMotion Manager and selecting Scan devices.

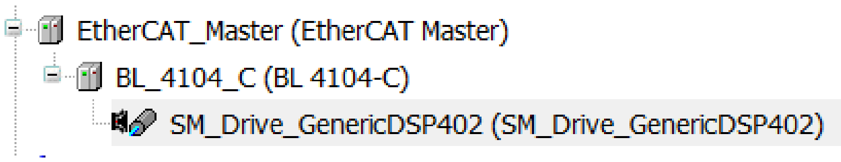


All servo drives connected to the fieldbus are detected and can be added to the project by clicking the Copy to project button. Afterwards the selected servo drives are displayed as devices connected to SoftMotion Manager.

Set PDO configuration

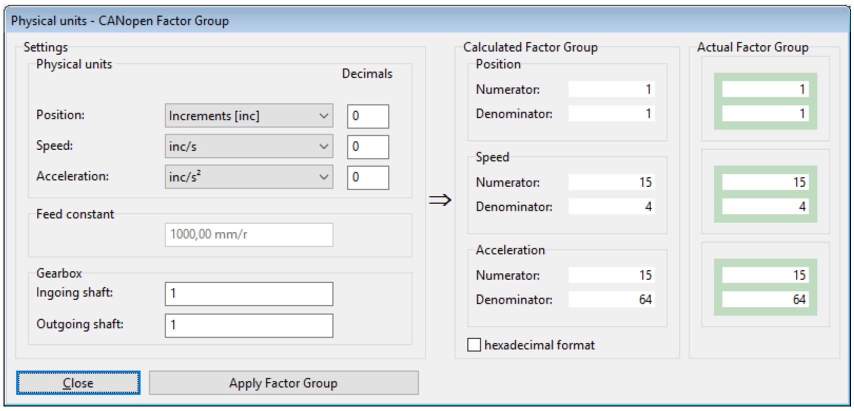
In contrast to CANopen, the complete PDO configuration of the cyclic data is done automatically via the ESI XML file, so that a SoftMotion axis can now be added directly to the servo drive.

Right click on the BL 4104-C to add a DSP402-compatible SoftMotion axis:



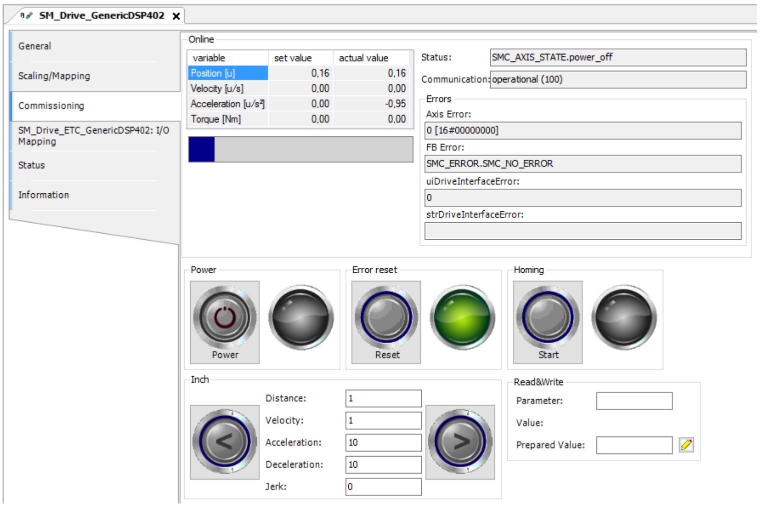
Adjust conversion factors

To ensure that the units of the cyclically exchanged values (e.g. for position and speed) match between master and servo drive, the scaling in the servo drive must be adapted to the scaling in the master. Therefore the following scaling should be set in the MPC ServoCommander under Parameters/Field bus/CANopen/Display units:



With this scaling a maximum of +-32768 revolutions with 16 bit resolution can be displayed on the bus. If this is not sufficient, the scaling of the setpoints transmitted on the bus can be adapted. This is described in section 3.3 *Factor Group* on page 43.

If the scaling is configured correctly, the actual position values should now be displayed in the Commissioning window of the CoDeSys SoftMotion axis:



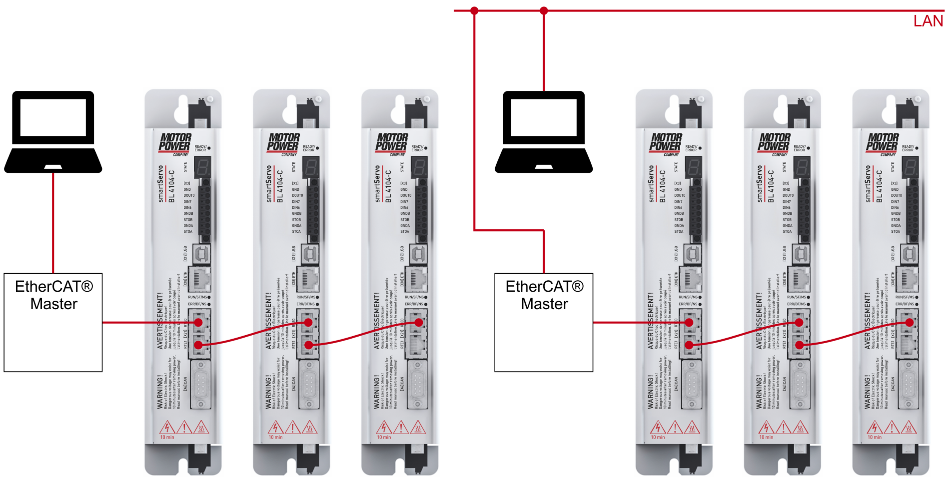
The axis can now be moved from the Commissioning tab for testing. In addition, the axis is now ready for implementation in the PLC project. A detailed description of all parameters of the servo drive and the implemented operating modes can be found from section 3 *Parameterisation* on page 36.

2.2.7 EoE (Ethernet over EtherCAT®)

Servo drives of the BL 4000 series support the EoE profile (Ethernet over EtherCAT®). In this case, normal Ethernet packets are also routed by the Ethernet master via the EtherCAT® network. This enables the MPC ServoCommander to establish Ethernet communication with the servo drives in the EtherCAT® network without additional cabling of the LAN interfaces.

EoE does not have to be activated separately in the servo drive, but only configured in the EtherCAT® master.

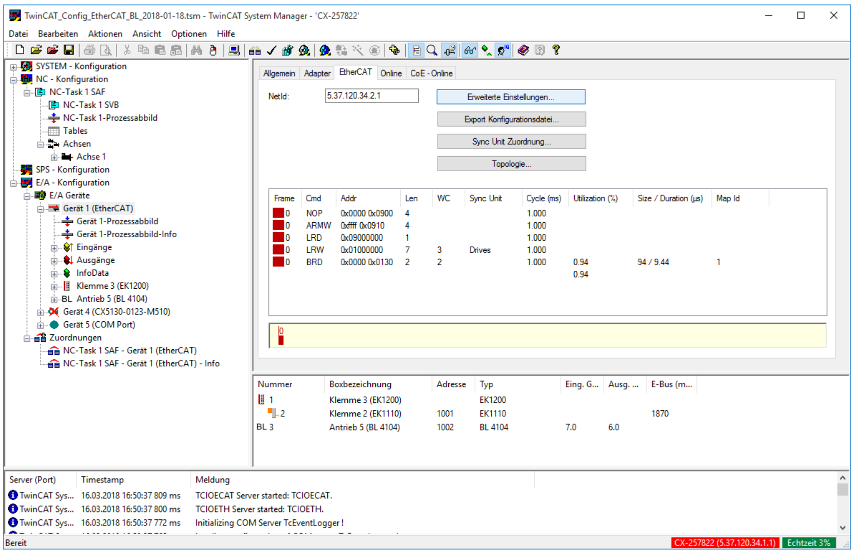
There are two different connection options for the EoE profile. In the first case, the laptop/PC running the MPC ServoCommander is connected directly to the controller, in the second case both are operated on a common LAN.



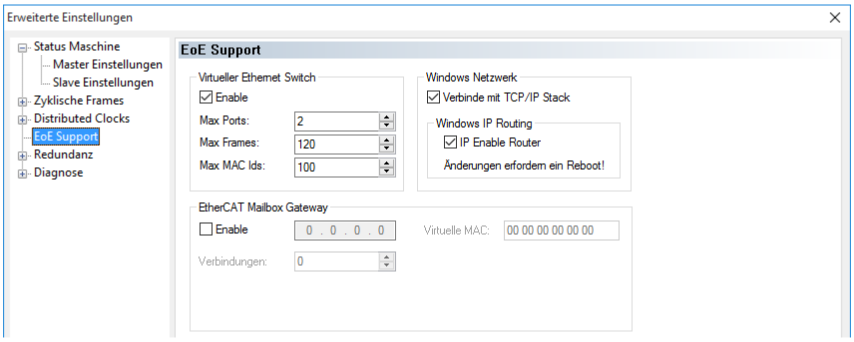
2.2.7.1 Activating EoE in the master

The activation of the EoE function is explained below using the example of a Beckhoff controller. The example assumes that an EtherCAT® network already exists and that cyclic communication with the drives is possible.

Select Device 1 (EtherCAT®) in the TwinCAT System Manager and click on Advanced Settings in the EtherCAT tab

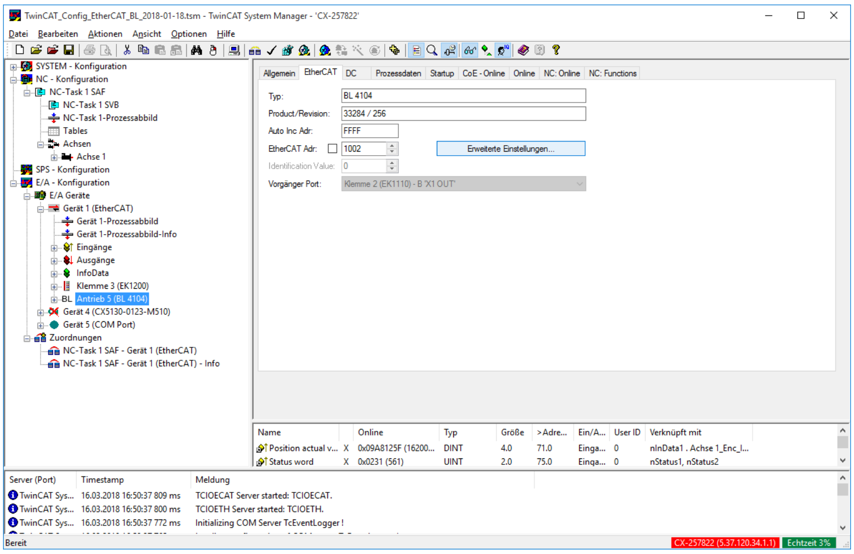


Select the entry EoE Support and activate Virtual Ethernet Switch and Connect to TCP/IP Stack. In the Windows IP Routing section, the IP Enable Router field must be selected. This enables the forwarding of standard Ethernet packets in the controller.

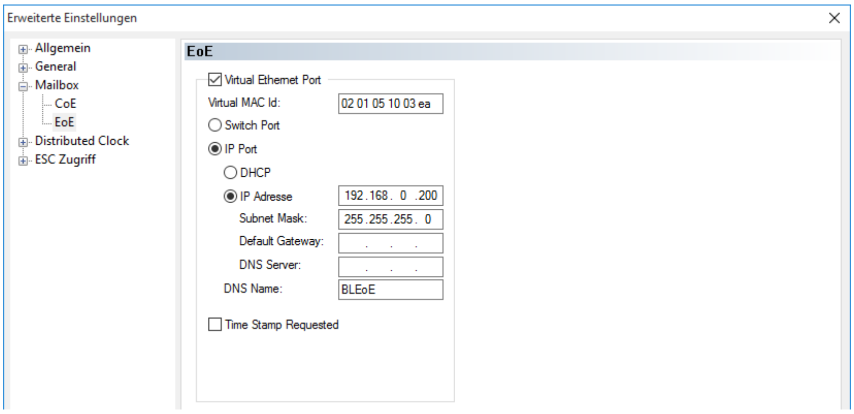


Finally, EoE support must be activated in the servo drive for each servo drive.

Select the corresponding drive, in this example Drive 5 (BL 4104) and click on Advanced Settings in the EtherCAT tab.



Click Mailbox / EoE, enable Virtual Ethernet Port and select IP Port. At this point you have the choice whether you want to assign a fixed IP address to the device or whether it should be obtained dynamically via DHCP. This requires that a corresponding DHCP server is located in the network.



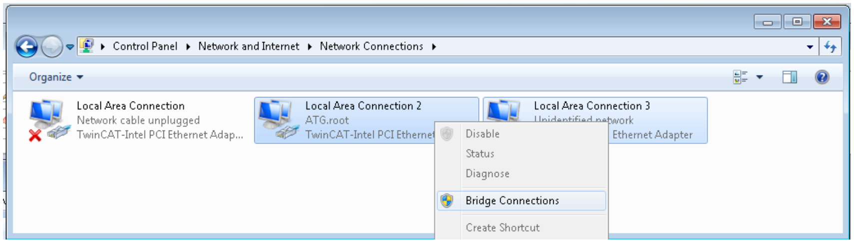
Finally, the new configuration must be loaded and activated on the controller. The servo drive is now displayed in the device search of the MPC ServoCommander as if the servo drive is connected directly via the Ethernet parameterisation interface (X18). If this is not the case, a "bridge" must also be activated within the Beckhoff controller. This is described in the following chapter.

2.2.7.2 Configure Bridge

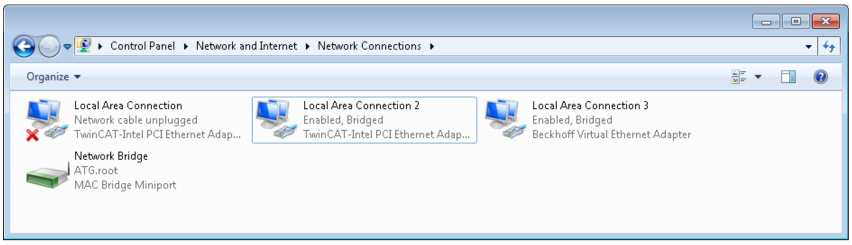
To make this setting, you must log in directly to the operating system of the Beckhoff controller.

Select Network and Internet in the Control Panel. Select the appropriate Ethernet connections (in our case TwinCAT Intel PCI Ethernet Adapter and Beckhoff Virtual Ethernet Adapter).

Press the right mouse button and select Bridge Connections.



Afterwards a Network Bridge is displayed.



3 Parameterisation

Before the servo drive can perform the desired task (torque control, speed control, positioning), numerous parameters of the servo drive must be adapted to the motor used and the specific application. This can be done either via the MPC ServoCommander or via CANopen.

The order in which the parameters are set can be based on the order of the following chapters. If the servo drive is already fully parameterised, you can continue directly with section 4 *Device Control* on page 102 or section 5 *Operating modes* on page 122.

INFORMATION **Seven-segment display of the servocontroller shows an "A"**

Servo drives with a seven-segment display show an "A" (Attention) if the servo drive has not yet been parameterised. If the servo drive is to be parameterised completely via CANopen, you must write to object $6510_h\_C0_h$ to suppress this display. (See section 3.17.1.16 *Object 6510h_C0h: commissioning_state* on page 99).

In addition to the parameters described here in detail, the object directory of the servo drive contains further parameters that must be implemented according to CANopen. They usually do not contain any information that can be used meaningfully with our servo drives. If required, the specification of such objects can be found in the corresponding standards (see section 7.1 *CANopen* on page 185).

Description of the parameters

All parameters of the drive are described in a uniform way. If the parameter is a simple data type (VAR), it is described as follows:

Index	**Index (hexadecimal)**			
Name	**Name of the parameter**			
Info	Unit	rw	PDO	Data type
Value	Value range	Default value		

If the parameter is a structured data type (ARRAY/RECORD), it is described as follows:

Index	**Index (hexadecimal)**	
Name	**Name of the parameter group**	
Type	Object code	Max

Sub-Index	**Subindex (hexadecimal)**			
Name	**Name of the parameter**			
Info	Unit	rw	~~PDO~~	Data type
Value	Value range	Default value		

The individual fields have the following meaning:

Field	Meaning
Index (hexadecimal)	The main index of the described parameter.
Subindex (hexadecimal)	The subindex of the described parameter. If this is not specified, the subindex is zero.
Name of the parameter group	Plain text name of the parameter group.
Name of the parameter	Plain text name of the parameter.
Object code	Specifies whether the data type is simple or structured: • VAR: Simple data type • ARRAY: Group of parameters that all have the same data type. • RECORD: Group of parameters that have different data types.
Max	Maximum subindex of the group.
Data type	Data type of the parameter or the ARRAY: A list of the supported data types can be found in section 6.2 *Access via SDO* on page 161.
Unit	Physical unit of the parameter.
Access	Specifies whether the parameter may be read (ro), written (wr) or read and written (rw).
PDO ~~PDO~~	Specifies whether the parameter may be mapped into a PDO.
Value range	The range of permissible values for this parameter.
Default value	Value that is effective on factory setting or after successful writing to *3.1.2.1 Object 1011h: restore_default_parameters*.

3.1 Loading and saving parameter sets

3.1.1 Overview

The servo drive has three parameter sets:

Current parameter set

This parameter set is located in the servo drive's volatile memory (RAM) and contains the parameters that are currently in use. It can be read and written as required with the parameterization program MPC ServoCommander or via the CAN bus. When the servo drive is switched on, the **Application parameter set** is copied to the **Current parameter set**.

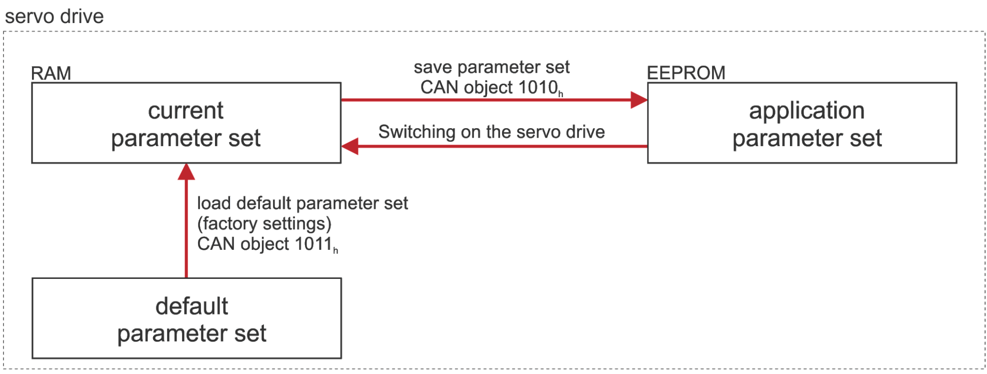
Application parameter set

The **Current parameter set** can be saved in the non-volatile flash memory (EEPROM) so that it is available again after the next power-up. The saving process is triggered with a write access to the CANopen object 1010_h_01_h (save_all_parameters).

Default parameter set

This is the unchangeable parameter set of the servo drive specified by the manufacturer as standard. The **Default parameter set** can be copied to the **Current parameter set** by a write operation to the CANopen object 1011_h_01_h (restore_all_default_parameters). This copying process is only possible when the power stage is switched off.

The following diagram illustrates the relationships between the individual parameter sets.



Two different concepts for parameter set management are conceivable:

Concept 1: The parameter set is created with the MPC ServoCommander and also transferred completely to the individual servo drives with the MPC ServoCommander. Using this method, only the objects that are exclusively accessible via CANopen must be set via the CAN bus. The disadvantage here is that the parameterization software is required for each commissioning of a new machine or in the event of a repair (servo drive replacement).

Concept 2: This variant is based on the fact that most application-specific parameter sets differ from the default parameter set only in a few parameters. This makes it possible to rebuild the **Current parameter set** each time the system is switched on via the CAN bus. For this purpose, the master controller first loads the **Default parameter set** by calling the CANopen object 1011_h_01_h (restore_all_default_parameters). Then only the deviating objects are transferred, which is very fast due to the small number of objects. An advantage is that this procedure also works with unparameterised servo drives, so that commissioning new systems or replacing individual servo drives is unproblematic and the parameterization software MPC ServoCommander is not required for this purpose.

⚠ CAUTION Risk of injury due to incorrectly parameterised servo drive

An incorrectly parameterised servo drive can cause uncontrolled rotary movements and thus personal injury or damage to property.

Before switching on the power stage for the very first time, make sure that the servo drive contains the desired parameters.

3.1.2 Description of objects

3.1.2.1 Object 1011_h: restore_default_parameters

Index	**1011_h**	
Name	**restore_parameters**	
Type	ARRAY	01_h

Sub-Index	**01_h**			
Name	**restore_all_default_parameters**			
Info	--	rw	~~PDO~~	UINT32
Value	$64616F6C_h$ („load“), 1 (read access)	--		

The object 1011_h_01_h (restore_all_default_parameters) allows the **Current parameter set** to be set to a defined state. To do this, the **Default parameter set** is copied into the **Current parameter set**. The copying process is triggered when "load" is written in hexadecimal form to this object.

This command is only executed when the output stage is deactivated. Otherwise the SDO error *08 00 00 22h* is issued. If the wrong identifier is sent, error *08 00 00 20h* is issued. If the object is accessed in read mode, a 1 is returned to indicate that resetting to default values is supported.

3.1.2.2 Object 1010_h: store_parameters

Index	**1010_h**			
Name	**store_parameters**			
Type	ARRAY			01_h

Sub-Index	**01_h**			
Name	**save_all_parameters**			
Info	--	rw	~~PDO~~	UINT32
Value	65766173_h („save“), 1 (read access)	--		

If the **Default parameter set** is also to be saved as the **Application parameter set**, 1010_h_01_h (save_all_parameters must be called in addition.

If the object is written via an SDO, the default behavior is that the SDO is answered immediately. The response therefore does not reflect the end of the saving process. However, the behavior can be changed using object 6510_h_$F0_h$ (compatibility_control).

3.2 Compatibility settings

3.2.1 Overview

The object compatibility_control has been introduced in order to remain compatible with earlier device series on the one hand, and to be able to carry out changes and corrections compared to the DSP402 and DS301 on the other. In the default parameter set, this object returns 0, that is, compatibility with earlier versions. For new applications, we recommend that you set the defined bits to ensure the highest possible level of compliance with the standards mentioned.

3.2.2 Description of objects

3.2.2.1 Object 6510_h_$F0_h$: compatibility_control

Index	**6510_h**			
Name	**drive_data**			
Type	RECORD			$F0_h$

Sub-Index	**$F0_h$**			
Name	**compatibility_control**			
Info	--	rw	~~PDO~~	UINT16
Value	0...7FFh, see Table	--		

Bit	Name	Value	Description
Bit 0	`homing_method_scheme*`	0001_h	The bit has the same meaning as bit 2 and is present for compatibility reasons. If bit 2 is set, this bit is also set and vice versa.
Bit 1	`reserved`	0002_h	The bit is reserved. It must not be set.
Bit 2	`homing_method_scheme`	0004_h	If this bit is set, the homing methods 32... 35 are numbered according to DSP402, otherwise the numbering is compatible with earlier implementations (see also section 5.2.3 *Homing sequences* on page 128). If this bit is set, bit 0 is also set and vice versa.
Bit 3	`reserved`	0008_h	The bit is reserved. It must not be set.
Bit 4	`response_after_save`	0010_h	If this bit is set, the response to save_all_parameters is not sent until saving is complete. This can take several seconds, which may cause a timeout in the PLC. If the bit is cleared, the response is sent immediately, but it must be taken into account that the saving process is not yet complete.
Bit 5	`reserved`	0020_h	The bit is reserved. It must not be set.

Bit	Name	Value	Description
Bit 6	homing_to_zero	0040_h	When using CANopen, the homing run consists of only 2 phases (search run and crawl run). The drive does NOT move to the determined zero position (which may be shifted to the found reference position, e.g. by the homing_offset). If this bit is set, the option selected in the MPC ServoCommander under Go to zero position after homing is used. In addition, the value given under max. homing distance permitted is used for the maximum search distance of the reference run. See section 5.2 *Homing Mode* on page 124.
Bit 7	device_control	0080_h	If this bit is set, bit 4 of the statusword (voltage_enabled) is output according to DSP 402 v2.0. In addition, the FAULT_ REACTION_ ACTIVE state can be distinguished from the FAULT state. See section 4 *Device Control* on page 102.
Bit 8	reserved	0100_h	The bit is reserved. It must not be set.
Bit 9	uzk_preload_ready	0200_h	If this bit is set, a set bit 4 (voltage_enabled) in the statuswordindicates that the DC link is fully loaded. If this bit is cleared, bit 4 indicates that the output stage is switched on. See section 4 *Device Control* on page 102.
Bit 10	home_offset_sign	0400_h	If this bit is set, the home_offset($607C_h$) is subtracted from the reference position instead of added, so that the drive is at the home_ offset position (instead of -home_offset) after the reference run.

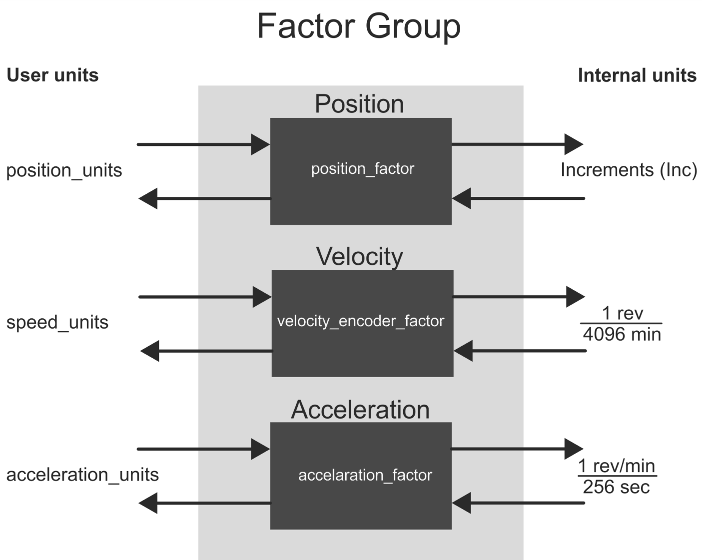
3.3 Factor Group

3.3.1 Overview

Usually, the values transmitted via the CAN bus are converted by the controller in such a way that they match the application used. If this is not the case, the scaling of the values transmitted on the bus can be adjusted directly using the Factor Group.

This may also be necessary if the resolution of the values transmitted on the bus is not sufficient, e.g. because the standard settings only allow a differentiation of +-32768 revolutions.

The servo drive converts the read or written values into its internal units with the help of the Factor Group. For each physical quantity (position, velocity and acceleration) a conversion factor is available to adapt the user units to the own application. The units set by the Factor Group are generally referred to as position_unit, speed_unit or acceleration_unit. The following figure illustrates the function of the Factor Group:



All parameters in the servo drive are always stored in internal units and are only converted by means of the Factor Group when writing or reading.

For this reason, the Factor Group should be set before the very first parameterisation and should not be changed afterwards.

By default, the Factor Group is set to the following units:

Quantity	Designation	Unit	Description
Length	position_unit	Increments	65536 increments per revolution
Speed	speed_unit	min^{-1}	Revolution per minute
Acceleration	acceleration_unit	$(min^{-1})/s$	Speed increase in revolutions per minute per second

3.3.2 Parameterisation of the Factor Group

The Factor Group can be conveniently set via the MPC ServoCommander:
Parameters/Field bus/CANopen/Display units or
Parameters/Feld bus/Ethercat/Display units

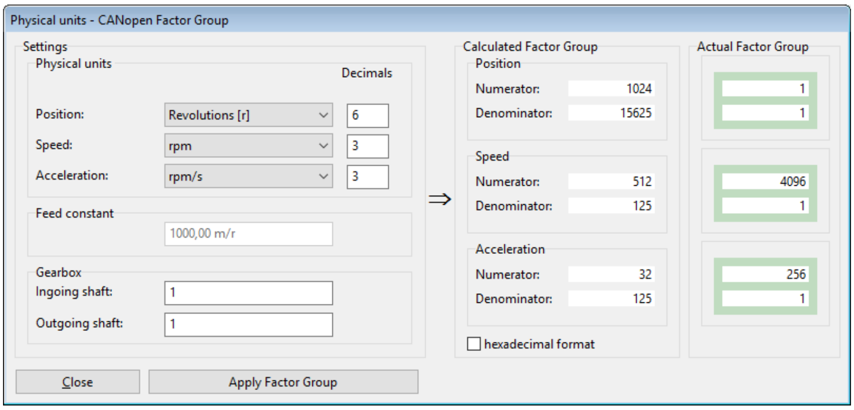


Figure 5: "CANopen Factor Group" window

Under Settings/Physical Units the desired unit for the position values (Position), Speed and Acceleration can be selected separately. In addition, the desired number of decimal places (Decimals) and a gear (Gearbox) can be included.

If a length unit is selected as the position unit, the Feed constant can also be specified.

The results of the setting selected in this way are displayed under Calculated Factor Group and can be transferred to the servo drive by clicking the Apply Factor Group button.

3.3.3 Description of objects

3.3.3.1 Object 6093h: position_factor

The object position_factor is used to convert all length units of the application from position_unit to the internal unit **increments** (65536 increments correspond to 1 revolution). It consists of numerator and denominator. The position_factor must not be greater than 2^{24}.

Index	**6093h**			
Name	**position_factor**			
Type	ARRAY			02h

Sub-Index	**01h**			
Name	**numerator**			
Info	--	rw	PDO	UINT32
Value	--	1		

Sub-Index	**02h**			
Name	**divisor**			
Info	--	rw	PDO	UINT32
Value	--	1		

3.3.3.2 Object 6094h: velocity_encoder_factor

The object velocity_encoder_factor is used to convert all velocity values of the application from speed_unit to the internal unit **revolutions per 4096 minutes**. It consists of numerator and denominator.

Index	**6094h**			
Name	**velocity_encoder_factor**			
Type	ARRAY			02h

Sub-Index	**01h**			
Name	**numerator**			
Info	--	rw	PDO	UINT32
Value	--	1000h		

Sub-Index	**02h**			
Name	**divisor**			
Info	--	rw	PDO	UINT32
Value	--	1		

3.3.3.3 Object 6097$_h$: acceleration_factor

The object acceleration_factor is used to convert all acceleration values of the application from acceleration_unit to the internal unit **revolutions per minute per 256 seconds**. It consists of numerator and denominator.

Index	**6097$_h$**			
Name	**acceleration_factor**			
Type	ARRAY			02$_h$

Sub-Index	**01$_h$**			
Name	**numerator**			
Info	--	rw	PDO	UINT32
Value	--	100$_h$		

Sub-Index	**02$_h$**			
Name	**divisor**			
Info	--	rw	PDO	UINT32
Value	--	1		

3.3.3.4 Object 607E$_h$: polarity

The sign of the position and velocity values of the servo drive can be set with the polarity object. This can be used to invert the direction of rotation of the motor with identical setpoints.

In most applications it is useful to set the position_polarity_flag and the velocity_polarity_flag to the same value.

Setting the position_polarity_flag or the velocity_polarity_flag only affects parameters during reading and writing. Parameters already present in the servo drive are not changed.

Index	**607E$_h$**			
Name	**polarity**			
Info	--	rw	PDO	UINT8
Value	0, 40$_h$, 80$_h$, C0$_h$	--		

Bit	Value	Name	Description
6	40$_h$	`velocity_polarity_flag`	0:multiply by 1 (default) 1:multiply by –1 (invers)
7	80$_h$	`position_polarity_flag`	0:multiply by 1 (default) 1:multiply by –1 (invers)

3.4 Power stage parameters

3.4.1 Overview

The DC link is supplied with mains voltage via a pre-charging circuit. This limits the current and controls the charging process. The precharge control is bypassed when the DC link is fully charged. This state is a condition for enabling the servo drive. The rectified mains voltage is smoothed with the capacitors of the DC link. The motor is supplied from the DC link via the IGBTs.

The output stage contains a number of monitoring functions, some of which can be parameterised:

- Controller enable logic (software- and hardware enable)
- Overvoltage / undervoltage monitoring of the DC link
- Overcurrent monitoring
- Power section monitoring

3.4.2 Description of objects

3.4.2.1 Object $6510_h\_10_h$: enable_logic

To enable the power stage of the servo drive, the digital input Controller enable must be set: The Controller enable is processed by the microcontroller of the servo drive. Depending on the operating mode, the servo drive reacts differently after this signal is disabled:

› Positioning mode and speed-controlled operation

After resetting the signal, the motor is braked with a defined braking ramp. The output stage is only switched off when the motor speed is below 10 min^{-1} and the holding brake, if present, has been applied.

› Torque-controlled operation

The output stage is switched off immediately after the signal is reset. At the same time a possibly existing holding brake is applied. The motor coasts down unbraked or is only stopped by a possibly existing holding brake.

⚠ DANGER ⚠ Danger to life due to electric shock!

Removing the Controller enable does not guarantee that the motor is voltage-free.

When operating the servo drive via CAN or EtherCAT, the digital input Controller enable can be connected to 24V and the enable controlled via the bus. To do this, object 6510_h_10_h (enable_logic) must be set to 2 (for CAN) or 8 (for EtherCAT). For safety reasons, this is done automatically when the fieldbus is activated (even after a reset of the servo drive).

Index	**6510_h**				
Name	**drive_data**				
Type	RECORD				$F0_h$

Sub-Index	**10_h**			
Name	**enable_logic**			
Info	--	rw	~~PDO~~	UINT16
Value	$0...18_h$	--		

Value	Description
0	Digital Input DIN5
1_h	DIN5 + Parameterisation interface
2_h	DIN5 + CAN
3_h	DIN5 + PROFIBUS/PROFINET
8_h	DIN5 + EtherCAT
11_h	Parameterisation interface only
12_h	CAN bus only
13_h	PROFIBUS/PROFINET only
18_h	EtherCAT only

3.4.2.2 Object 6510_h_30_h: pwm_frequency

The switching losses of the output stage are proportional to the switching frequency of the power transistors. Some servo drives can draw a little more power by halving the normal PWM frequency. However, this increases the current ripple caused by the output stage. Switching is only possible when the output stage is switched off.

Index	**6510_h**				
Name	**drive_data**				
Type	RECORD				$F0_h$

Sub-Index	**30_h**			
Name	**pwm_frequency**			
Info	--	rw	~~PDO~~	UINT16
Value	0, 1	0		

Value	Description
0	Standard power stage frequency
1	Half power stage frequency

3.4.2.3 Object $6510_h\_3A_h$: enable_enhanced_modulation

With the object enable_enhanced_modulation the enhanced sine modulation can be activated. It allows for a better utilization of the DC bus voltage and thus about 14% higher speeds. The disadvantage is that the control behavior and the smooth running of the motor is slightly worse at very low speeds. The parameter may only be changed with the power stage switched off and only becomes effective after a reset. To do this, the parameter set must first be saved (save_all_parameters).

Index	**6510_h**	
Name	**drive_data**	
Type	RECORD	$F0_h$

Sub-Index	**$3A_h$**			
Name	**enable_enhanced_modulation**			
Info	--	rw	~~PDO~~	UINT16
Value	0, 1	0		

Value	Description
0	Enhanced sine modulation OFF
1	Enhanced sine modulation ON

3.4.2.4 Object $6510_h\_31_h$: power_stage_temperature

The temperature of the power stage can be read out via the object power_stage_temperature. If the temperature specified in object $6510_h\_32_h$ (max_power_stage_temperature) is exceeded, the power stage switches off and an error message is issued.

Index	**6510_h**	
Name	**drive_data**	
Type	RECORD	$F0_h$

Sub-Index	**31_h**			
Name	**power_stage_temperature**			
Info	°C	ro	PDO	INT16
Value	--	--		

3.4.2.5 Object 6510_h_32_h: max_power_stage_temperature

The temperature of the power stage can be read out via the object 6510_h_31_h (power_stage_temperature). If the temperature specified in object max_power_stage_temperature is exceeded, the power stage switches off and an error message is issued.

Index	**6510_h**	
Name	**drive_data**	
Type	RECORD	$F0_h$

Sub-Index	**32_h**			
Name	**max_power_stage_temperature**			
Info	°C	ro	~~PDO~~	INT16
Value	--	--		

Device type	Value
BL 4102-C	85°C
BL 4104-C	85°C
BL 4304-C	90°C
BL 4308-C	85°C
BL 4312-C	75°C

3.4.2.6 Object 6510_h_33_h: nominal_dc_link_circuit_voltage

Via the object nominal_dc_link_circuit_voltage the device nominal voltage can be read out in millivolts.

Index	**6510_h**	
Name	**drive_data**	
Type	RECORD	$F0_h$

Sub-Index	**33_h**			
Name	**nominal_dc_link_circuit_voltage**			
Info	mV	ro	~~PDO~~	UINT32
Value	--	--		

Device type	Value
BL 4102-C	325000
BL 4104-C	325000
BL 4304-C	560000
BL 4308-C	560000
BL 4312-C	560000

3.4.2.7 Object 6510_h_34_h: actual_dc_link_circuit_voltage

The object actual_dc_link_circuit_voltage can be used to read the current voltage of the DC link in millivolts.

Index	**6510_h**	
Name	**drive_data**	
Type	RECORD	$F0_h$

Sub-Index	**34_h**			
Name	**actual_dc_link_circuit_voltage**			
Info	mV	ro	PDO	UINT32
Value	--	--		

3.4.2.8 Object 6510_h_35_h: max_dc_link_circuit_voltage

The object max_dc_link_circuit_voltage specifies the DC link voltage at which the output stage is immediately switched off for safety reasons and an error message is sent.

Index	**6510_h**	
Name	**drive_data**	
Type	RECORD	$F0_h$

Sub-Index	**35_h**			
Name	**max_dc_link_circuit_voltage**			
Info	mV	ro	~~PDO~~	UINT32
Value	--	--		

Device type	Value
BL 4102-C	439979
BL 4104-C	439979
BL 4304-C	799976
BL 4308-C	799976
BL 4312-C	799976

3.4.2.9 Object $6510_h\_36_h$: min_dc_link_circuit_voltage

The servo drive has an undervoltage monitor. This can be activated via object $6510_h\_37_h$ (enable_dc_link_undervoltage_error). Object $6510_h\_36_h$ (min_dc_link_circuit_voltage) specifies the minimum DC link voltage. Below this voltage, error E 02-0 is raised.

Index	**6410_h**			
Name	**motor_data**			
Type	RECORD			14_h

Sub-Index	**36_h**			
Name	**min_dc_link_circuit_voltage**			
Info	mV	rw	~~PDO~~	UINT32
Value	0...1000000	--		

3.4.2.10 Object $6510_h\_37_h$: enable_dc_link_undervoltage_error

The undervoltage monitoring can be activated with the object enable_dc_link_undervoltage_error. The undervoltage monitoring can be activated with the object enable_dc_link_undervoltage_error. Object $6510_h\_36_h$ (min_dc_link_circuit_voltage) defines the DC link voltage below which an error is raised.

Index	**6510_h**			
Name	**drive_data**			
Type	RECORD			$F0_h$

Sub-Index	**37_h**			
Name	**enable_dc_link_undervoltage_error**			
Info	--	rw	~~PDO~~	UINT16
Value	0, 1	0		

Value	Description
0	Undervoltage error OFF (reaction Warning)
1	Undervoltage error ON (reaction Disable servo drive)

If this object is written, the error reaction of error 02-0 is modified. If 0 is written, the error reaction Warning is set. If 1 is written, the error reaction Disable servo drive is set. If the object is read, the reaction Disable servo drive or higher is reported as 1, all other error reactions as 0. See also section 3.18 *Error management* on page 100.

3.4.2.11 Object 6510_h_40_h: nominal_current

The nominal_current object can be used to read the nominal device current. This is the upper limit value which can be written into the object 6075_h (motor_rated_current). Due to a power derating, different values may be read depending on the servo drive cycle time and the power stage clock frequency.

Index	**6510_h**			
Name	**drive_data**			
Type	RECORD			$F0_h$
Sub-Index	**40_h**			
Name	**nominal_current**			
Info	mA	ro	~~PDO~~	UINT32
Value	--	see Table		

Device type	Value
BL 4102-C	2000
BL 4104-C	4000
BL 4304-C	4000
BL 4308-C	8000
BL 4312-C	12000

3.4.2.12 Object 6510_h_41_h: peak_current

The peak_current object can be used to read the maximum device current. This is the upper limit value which can be written into the object 6073_h (max_current). Due to a power derating, different values may be read depending on the servo drive cycle time and the power stage clock frequency.

Index	**6510_h**			
Name	**drive_data**			
Type	RECORD			$F0_h$
Sub-Index	**41_h**			
Name	**peak_current**			
Info	mA	ro	~~PDO~~	UINT32
Value	--	see Table		

Device type	Value
BL 4102-C	6400
BL 4104-C	12800
BL 4304-C	12000
BL 4308-C	24000
BL 4312-C	30000

3.5 Current controller and motor adaption

NOTICE Damage to property due to incorrect settings

Incorrect settings of the current controller parameters and the current limits can destroy the motor and possibly also the servo drive within a very short time.

3.5.1 Overview

⚠ CAUTION Danger of injury due to dangerous movements

If the phase order of the motor or angle encoder cable is twisted, positive feedback may occur, which means that the speed in the motor cannot be controlled. The motor can rotate uncontrolled.

The parameter set of the servo drive must be adapted for the connected motor and the cable set used. The following parameters are affected:

- Rated current (depending on the motor)
- Overload capacity (depending on the motor)
- Number of poles (depending on the motor)
- Current controller (depending on the motor)
- Direction of rotation (depending on the motor and the phase sequence in the motor and angle encoder cable)
- Offset angle (depending on motor and phase sequence in motor and angle encoder cable)

These data must be determined with the program MPC ServoCommander when a motor type is used for the first time. For a number of motors you can also obtain ready-made parameter sets from your dealer. Please note that the direction of rotation and offset angle also depend on the cable set used. The parameter sets therefore only work with identical wiring.

3.5.2 Description of objects

3.5.2.1 Object 6075_h: motor_rated_current

This value can be taken from the motor nameplate and is entered as an effective value (RMS) in the unit milliampere. No current can be entered which is above the servo drive rated current (6510_h_40_h, nominal_current).

Index	**6075_h**			
Name	**motor_rated_current**			
Info	mA	rw	PDO	UINT32
Value	0...nominal_current	--		

INFORMATION **Objects not independent**

If object 6075_h (motor_rated_current) is written with a new value, object 6073_h (max_current) must also be reparameterised in any case.

3.5.2.2 Object 6073_h: max_current

Servo motors may normally be overloaded for a certain period of time. This object is used to set the maximum permissible motor current. It refers to the rated motor current (Object 6075_h, motor_rated_current) and is set in thousandths. The value range is limited upwards by the maximum servo drive current (Object 6510_h_41_h, peak_current). Many motors can be overloaded by a factor of 2 for a short time. In this case, the value 2000 must be written into this object. Object 6073_h (max_current) may only be written to after Object 6075_h (motor_rated_current) has previously been written with a valid value.

Index	**6073_h**			
Name	**max_current**			
Info	‰ (1000 = motor_rated current)	rw	PDO	UINT16
Value	--	--		

3.5.2.3 Object $604D_h$: pole_number

The number of poles of the motor can be taken from the motor data sheet or the parameterization program MPC ServoCommander. The number of poles is always even. Often the number of pole pairs is given instead of the number of poles. In this case, the number of poles is twice the number of pole pairs. This object is not changed by restore_default_parameters, but it can be reset by selecting File / Parameter set / Load default servo parameter set.

Index	**$604D_h$**			
Name	**pole_number**			
Info	--	rw	PDO	UINT8
Value	2...254	--		

3.5.2.4 Object 6410_h_11_h: encoder_offset_angle

The servo motors used have permanent magnets on the rotor. These generate a magnetic field whose orientation to the stator depends on the rotor position. For electronic commutation, the servo drive must always adjust the electromagnetic field of the stator at the correct angle to this permanent magnetic field. To do this, it continuously determines the rotor position with an angle encoder (resolver etc.).

The orientation of the angle encoder to the permanent magnetic field must be entered in the object encoder_offset_angle. This angle can be determined with the parameterisation program MPC ServoCommander (Parameters / Device parameters / Encoder / Settings).

The angle determined with the MPC ServoCommander is in the range of ±180°. It must be converted as follows:

encoder_offset_angle = Offset of encoder × 32767 / 180°

This object is not changed by restore_default_parameters, but it can be reset by selecting File / Parameter set / Load default servo parameter set.

Index	**6410$_h$**			
Name	**motor_data**			
Type	RECORD			14$_h$
Sub-Index	**11$_h$**			
Name	**encoder_offset_angle**			
Info	180° / 32767	rw	PDO	INT16
Value	--	--		

3.5.2.5 Object 6410$_h$_10$_h$: phase_order

The phase_order object considers twists between motor cable and angle encoder cable. It can be taken from the parameterisation program MPC ServoCommander. This object is not changed by restore_default_parameters, but it can be reset by selecting File / Parameter set / Load default servo parameter set

Index	**6410$_h$**			
Name	**motor_data**			
Type	RECORD			14$_h$
Sub-Index	**10$_h$**			
Name	**phase_order**			
Info	--	rw	~~PDO~~	UINT16
Value	0, 1	0		

Value	Description
0	Right
1	Left

3.5.2.6 Object $6410_h\_03_h$: iit_time_motor

Servo motors may normally be overloaded for a certain period of time. This object is used to specify how long the connected motor may be operated with the current specified in object 6073_h (max_current). After the I²t time has expired, the current is automatically limited to the value specified in object 6075_h (motor_rated_current) to protect the motor. The default setting is two seconds and is applicable for most motors.

Index	**6410_h**	
Name	**motor_data**	
Type	RECORD	14_h

Sub-Index	**03_h**			
Name	**iit_time_motor**			
Info	ms	rw	~~PDO~~	UINT16
Value	0...10000	--		

3.5.2.7 Object $6410_h\_04_h$: iit_ratio_motor

The object iit_ratio_motor can be used to read the current I²t limitation of the motor in per mille.

Index	**6410_h**	
Name	**motor_data**	
Type	RECORD	14_h

Sub-Index	**04_h**			
Name	**iit_ratio_motor**			
Info	‰	ro	~~PDO~~	UINT16
Value	--	--		

3.5.2.8 Object $6510_h\_3D_h$: iit_ratio_servo

The object iit_ratio_servo can be used to read the current I²t limitation of the power stage in per mille.

Index	**6510_h**	
Name	**drive_data**	
Type	RECORD	$F0_h$

Sub-Index	**$3D_h$**			
Name	**iit_ratio_servo**			
Info	‰	ro	PDO	UINT16
Value	--	--		

3.5.2.9 Object $6510_h\_38_h$: iit_error_enable

The object iit_error_enable defines how the servo drive behaves when the I²t limitation occurs. Either this is only indicated in the statusword, or error E 31-0 is raised.

Index	**6510_h**			
Name	**drive_data**			
Type	RECORD			$F0_h$

Sub-Index	**38_h**			
Name	**iit_error_enable**			
Info	--	rw	~~PDO~~	UINT16
Value	0, 1	0		

Value	Description
0	I²t error OFF (Reaction Warning)
1	I²t error ON (Reaction Disable Servo Drive)

If this object is written, the error reaction of error 31-0 is modified. If 0 is written, the error reaction Warning is set. If 1 is written, the error reaction Disable servo drive is set. If the object is read, the reaction Disable servo drive or higher is reported as 1, all other error reactions as 0. See section 3.18 *Error management* on page 100.

3.5.2.10 Object $6510_h\_2E_h$: motor_temperature

This object can be used to read out the current motor temperature if an analog temperature sensor is connected. Otherwise, the value of the object is undefined.

Index	**6510_h**			
Name	**drive_data**			
Type	RECORD			$F0_h$

Sub-Index	**$2E_h$**			
Name	**motor_temperature**			
Info	°C	ro	PDO	INT16
Value	--	--		

3.5.2.11 Object 6410$_h$_14$_h$: motor_temperature_sensor_polarity

This object can be used to define whether a normally closed contact or a normally open contact is used as a digital motor temperature sensor.

Index	**6410$_h$**	
Name	**motor_data**	
Type	RECORD	14$_h$

Sub-Index	**14$_h$**			
Name	**motor_temperature_sensor_polarity**			
Info	--	rw	PDO	INT16
Value	0, 1	0		

Value	Description
0	Normally closed contact
1	Normally open contact

3.5.2.12 Object 6510$_h$_2F$_h$: max_motor_temperature

If the motor temperature defined in this object is exceeded, the reaction as set in the error management (error E 03-0, motor overtemperature analog) is executed. If a reaction is parameterised which leads to the drive being switched off, an emergency message is sent. For parameterisation of the error management, see section 3.18 *Error management* on page 100.

Index	**6510$_h$**	
Name	**drive_data**	
Type	RECORD	F0$_h$

Sub-Index	**2F$_h$**			
Name	**max_motor_temperature**			
Info	°C	rw	~~PDO~~	INT16
Value	20...300	--		

3.5.2.13 Object $60F6_h$: torque_control_parameters

The data of the current controller must be taken from the parameterisation program MPC ServoCommander. The following conversions must be observed:

The gain of the current controller must be multiplied by 256. For a gain of 1.5 in the "Current controller" menu of the parameterisation program MPC ServoCommander, the value 384 = 180_h must be written into the torque_control_gain object.

The time constant of the current controller is specified in milliseconds in the parameterisation program MPC ServoCommander. In order to be able to transfer this time constant into the torque_control_time object, it must first be converted into microseconds. For a specified time of 0.6 milliseconds, the value 600 must be entered into the torque_control_time object accordingly. The lower limit must not be smaller than the current cycle time of the current controller (see section 3.17.1.12 *Object 6510h_B0h: cycletime_current_controller* on page 97).

Index	**$60F6_h$**			
Name	**torque_control_parameters**			
Type	RECORD			02_h

Sub-Index	**01_h**			
Name	**torque_control_gain**			
Info	256 = „1“	rw	~~PDO~~	UINT16
Value	0...(32*256)	--		

Sub-Index	**02_h**			
Name	**torque_control_time**			
Info	µs	rw	~~PDO~~	UINT16
Value	104...64401	--		

3.5.2.14 Object $203A_h$: torque_feed_forward

Specifies the current feedforward factor. This is parameterised in 10^{-7} A per set acceleration. This allows an acceleration profile set via CANopen to be run and the current during acceleration to be recorded. The quotient of current and acceleration can then be written directly to this object.

Index	**$203A_h$**			
Name	**torque_feed_forward**			
Info	A / (rev/min/s)	rw	~~PDO~~	UINT32
Value	0...208	--		

3.6 Velocity controller

3.6.1 Overview

NOTICE **Damage to property due to incorrect settings**

Incorrect settings of the controller parameters can lead to strong vibrations and possibly destroy parts of the machine.

The parameter set of the servo drive must be adapted for the application. Especially the gain is highly dependent on any masses coupled to the motor. The data must be optimally determined during commissioning of the system using the MPC ServoCommander parameterisation program.

3.6.2 Description of objects

3.6.2.1 Object $60F9_h$: velocity_control_parameters

The data of the speed controller can be taken from the parameterisation program MPC ServoCommander. The following conversions must be observed:

The gain of the speed controller must be multiplied by 256. For a gain of 1.5 in the "Speed controller" menu of the parameterisation program, the value 384 = 180_h must be written into the velocity_control_gain object.

The time constant of the speed controller is given in milliseconds in the parameterisation program. In order to be able to transfer this time constant into the object velocity_control_time, it must first be converted into microseconds. For a given time of 2.0 milliseconds, the value 2000 must be entered into the object velocity_control_time accordingly. The same applies to the object velocity_control_filter_time, with which the actual speed value filter is parameterised.

Index	**$60F9_h$**			
Name	**velocity_control_parameter_set**			
Type	RECORD			04_h

Sub-Index	**01_h**			
Name	**velocity_control_gain**			
Info	256 = „1“	rw	~~PDO~~	UINT16
Value	20...(64*256)	--		

Sub-Index	**02_h**			
Name	**velocity_control_time**			
Info	µs	rw	~~PDO~~	UINT16
Value	1...32000	--		

Sub-Index	**04_h**			
Name	**velocity_control_filter_time**			
Info	µs	rw	~~PDO~~	UINT16
Value	1...32000	--		

3.6.2.2 Object 2073_h: velocity_display_filter_time

The velocity_display_filter_time object can be used to set the filter time of the speed actual value filter, which filters the actual value for display.

Index	**2073_h**			
Name	**velocity_display_filter_time**			
Info	µs	rw	~~PDO~~	UINT32
Value	1000...50000	--		

INFORMATION **Object is also used for overspeed-protection**

Note that the object velocity_actual_value_filtered is used for the overspeed-protection. If the filter time is very long, a overspeed error is only detected after a respective delay.

3.7 Position Controller

3.7.1 Overview

This chapter describes all parameters required for the position controller. The position setpoint (position_demand_value) from the trajectory generator is applied to the input of the position controller. In addition, the actual position value (position_actual_value) is supplied by the angle encoder (resolver, incremental encoder etc.). The behavior of the position controller can be influenced by parameters. To keep the position control loop stable, a limitation of the output variable (control_effort) is possible. The output variable is fed into the speed controller as a speed setpoint value. All input and output variables of the position controller are converted by the Factor Group from the application-specific units into the respective internal units of the servo drive.

› Following error

The following_error_actual_value is the deviation of the actual position value (position_ actual_ value) from the position setpoint (position_demand_value). If this following error is larger than specified in the following_error_window for a certain period of time, bit 13 following_error is set in the statusword object. The permissible time period can be specified via the object following_error_time_out.

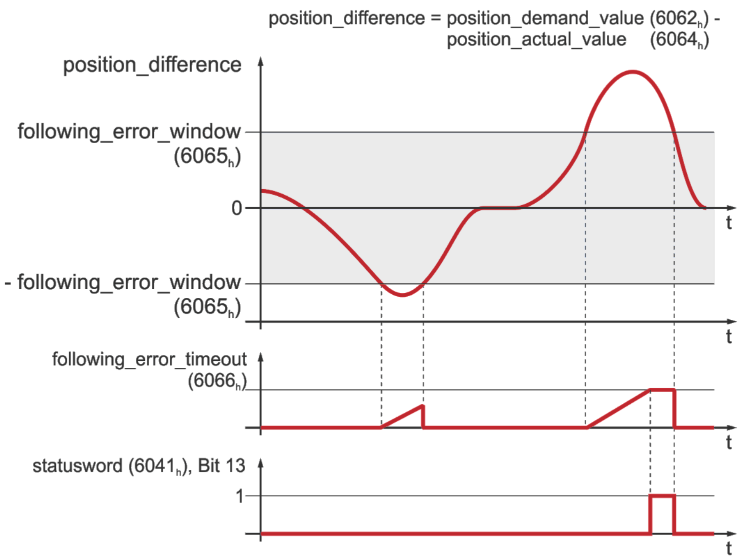


Figure 6: Following error – function overview

Figure 6 "Following error". It is monitored whether the difference between target position (position_demand_value) and actual position (position_actual_value) leaves the symmetrical following_error_window. If the position difference does not return to the window within a certain period of time (following_error_time_out), bit 13 in the statusword is set.

› Position reached (Target reached)

This function offers the possibility to define a position window around the target position. If the actual position of the drive is within this range for a certain time - the position_ window_time - the associated bit 10 (target_reached) is set in the statusword.

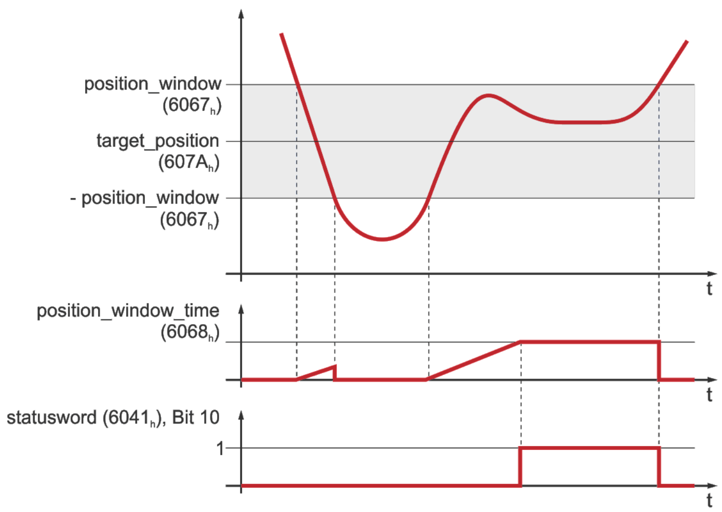


Figure 7: Position reached - function overview

Figure 7. It is monitored whether the actual position (position_actual_value) is within the symmetrical target position window (target_position+position_window, target_position-position_window). If the actual position remains in the target window longer than the waiting time (target_window_time) and the positioning is completed, bit 10 in the statusword is set.

3.7.2 Description of objects

3.7.2.1 Object $60FB_h$: position_control_parameter_set

The parameter set of the servo drive must be adapted for the application. The data of the position controller must be optimally determined with the aid of the program MPC ServoCommander when the machine is commissioned.

NOTICE Damage to property due to incorrect settings

Incorrect settings of the controller parameters can lead to strong vibrations and possibly destroy parts of the machine.

The position controller compares the setpoint position with the actual position and forms a correction speed (object $60FA_h$: control_effort) from the difference - taking into account the gain and possibly the integrator - which is fed to the speed controller. The position controller is relatively slow compared to the current and speed controller. Therefore, the servo drive works internally with feedforward controls, so that the correction work for the position controller is minimised and the servo drive is quickly in the steady state. A proportional element is normally sufficient as position controller.

The position controller data can be taken from the parameterization program MPC ServoCommander. The following conversions must be observed: The gain of the position controller must be multiplied by 256. For a gain of 1.5 in the Position Controller window of the parameterization program, the value 384 must be written into the object position_control_gain.

Normally the position controller does not need an integrator. In this case, the value zero must be entered in the object position_control_time. Otherwise, the time constant of the position controller must be converted into microseconds. For a time of 4.0 milliseconds, the value 4000 must be entered in the object position_control_time accordingly. Since the position controller converts even the smallest position deviations into significant correction speeds, in the event of a brief malfunction (e.g. brief clamping of the system), this would result in very violent control reactions with very high correction speeds. This can be avoided if the output of the position controller is limited sensibly (e.g. 500 min^{-1}) via the object position_control_v_max.

The object position_error_tolerance_window can be used to define the size of a position deviation up to which the position controller does not act (dead band). This can be used for stabilization purposes, for example, if there is backlash in the system.

Index	**$60FB_h$**	
Name	**position_control_parameter_set**	
Type	RECORD	05_h

Sub-Index	**01_h**			
Name	**position_control_gain**			
Info	256 = „1“	rw	~~PDO~~	UINT16
Value	0...(64*256)	--		

Sub-Index	**02_h**			
Name	**position_control_time**			
Info	µs	rw	~~PDO~~	UINT16
Value	0	--		

Sub-Index	**04_h**			
Name	**position_control_v_max**			
Info	speed_unit	rw	~~PDO~~	UINT32
Value	0...131072 min^{-1}	--		

Sub-Index	**05_h**			
Name	**position_error_tolerance_window**			
Info	position_unit	rw	~~PDO~~	UINT32
Value	--	--		

3.7.2.2 Object 6062$_h$: position_demand_value

The current position setpoint can be read out via this object. This is fed into the position controller by the travel curve generator.

Index	**6062$_h$**			
Name	**position_demand_value**			
Info	position_unit	ro	PDO	INT32
Value	--	--		

3.7.2.3 Object 202D$_h$: position_demand_sync_value

This object can be used to read the position setpoint of the synchronisation encoder. This is defined by object 2022$_h$ synchronization_encoder_select.

Index	**202D$_h$**			
Name	**position_demand_sync_value**			
Info	position_unit	ro	~~PDO~~	INT32
Value	--	--		

3.7.2.4 Object 6064$_h$: position_actual_value

The actual position can be read out via this object. This is fed to the position controller from the angle encoder.

Index	**6064$_h$**			
Name	**position_actual_value**			
Info	position_unit	ro	PDO	INT32
Value	--	--		

3.7.2.5 Object 6066$_h$: following_error_time_out

If a following error - longer than defined in this object - occurs, the corresponding bit 13 following_error is set in the statusword.

Index	**6066$_h$**			
Name	**following_error_time_out**			
Info	ms	rw	PDO	UINT16
Value	0...27314	--		

3.7.2.6 Object 6065$_h$: following_error_window

The object following_error_window defines a symmetrical range around the position setpoint (position_demand_value). If the actual position value (position_actual_value) is outside the following_error_window, then a following error occurs and bit 13 in the statusword is set. The reasons below can cause a following error:

- The drive is blocked
- The positioning speed is too high
- The acceleration values are too high
- The object following_error_window has a value that is too small
- The position controller is not correctly parameterised

Index	**6065$_h$**			
Name	**following_error_window**			
Info	position_unit	rw	PDO	UINT32
Value	--	--		

3.7.2.7 Object 60F4$_h$: following_error_actual_value

The current difference between position_demand_value (6062$_h$) and position_actual_value (6064$_h$) can be read from this object.

Index	**60F4$_h$**			
Name	**following_error_actual_value**			
Info	position_unit	ro	PDO	INT32
Value	--	--		

3.7.2.8 Object 60FA$_h$: control_effort

The output value of the position controller can be read out via this object. This value is fed internally into the speed controller as setpoint value.

Index	**60FA$_h$**			
Name	**control_effort**			
Info	speed_unit	ro	PDO	INT32
Value	--	--		

3.7.2.9 Object 6410$_h$_0F$_h$: rotor_position

The rotor_position can be read out via the object in per mil of one revolution.

Index	**6410$_h$**			
Name	**motor_data**			
Type	RECORD			14$_h$

Sub-Index	**0F$_h$**			
Name	**rotor_position**			
Info	‰ (1000 = 1 rev)	ro	PDO	UINT16
Value	--	--		

3.7.2.10 Object 6067$_h$: position_window

The object position_window defines a symmetrical range around the target position. If the actual position value (position_actual_value) is within this range for a certain time, the target position is considered to be reached.

Index	**6067$_h$**			
Name	**position_window**			
Info	position_unit	rw	PDO	UINT32
Value	--	--		

3.7.2.11 Object 6068$_h$: position_window_time

If the actual position of the drive is within the positioning window (position_window) for as long as defined in this object, the corresponding bit 10 target_reached is set in the statusword.

Index	**6068$_h$**			
Name	**position_window_time**			
Info	ms	rw	PDO	UINT16
Value	--	--		

3.7.2.12 Object 6510_h_22_h: position_error_switch_off_limit

The maximum permissible deviation between the target and actual position can be entered in the object position_error_switch_off_limit. In contrast to the Following Error message above, the output stage is switched off immediately if this limit is exceeded and an error is raised. The motor thus coasts down unbraked (unless there is a holding brake).

Index	**6510_h**			
Name	**drive_data**			
Type	RECORD			$F0_h$

Sub-Index	**22_h**			
Name	**position_error_switch_off_limit**			
Info	position_unit	rw	~~PDO~~	UINT32
Value	--	--		

Value	Description
0	Switch-off limit following error OFF (Reaction No action)
> 0	Switch-off limit following error ON (Reaction Disable power stage immediately)

If this object is written, the error reaction of error 17-0 is modified. If 0 is written, the error reaction No action is set. If a value greater than 0 is written, the error reaction Disable power stage immediately is set. If the object is read, the reaction Disable power stage immediately is reported as 1, all other error reactions as 0. See also section 3.18 *Error management* on page 100.

3.7.2.13 Object 2030_h: set_position_absolute

The object set_position_absolute can be used to move the readable actual position without changing the physical position. The drive does not carry out any movement. If an absolute encoder system is connected, the position displacement is stored in the encoder, if the encoder system allows this. In this case, the position offset is therefore retained after a reset. This storage operation runs in the background independently of this object. All parameters belonging to the encoder memory are also stored with their current values.

Index	**2030_h**			
Name	**set_position_absolute**			
Info	position_unit	wo	~~PDO~~	INT32
Value	--	--		

3.7.2.14 Object 607D$_h$: software_position_limit

The object array software_position_limit contains two sub-parameters that limit the maximum positioning range. If the drive leaves this range in Profile Position Mode, error 40-0 (Negative SW limit switch reached) or 40-1 (Positive SW limit switch reached) is raised.

Index	**607D$_h$**			
Name	**software_position_limit**			
Type	ARRAY			02$_h$

Sub-Index	**01$_h$**			
Name	**min_position_limit**			
Info	position_unit	rw	PDO	INT32
Value	--	--		

Sub-Index	**02$_h$**			
Name	**max_position_limit**			
Info	position_unit	rw	PDO	INT32
Value	--	--		

3.7.2.15 Object 607B$_h$: position_range_limit

The object array position_range_limit contains two sub-parameters that limit the numerical range of the position values. If one of these limits is exceeded, the actual position value automatically overflows to the other limit. This enables the parameterisation of so-called rotary axes. The limits that should physically correspond to the same position must be specified, for example 0° and 360°.

To make these limits effective, a rotary axis mode must be selected via 6510$_h$_20$_h$ (position_range_limit_enable) .

Index	**607B$_h$**			
Name	**position_range_limit**			
Type	ARRAY			02$_h$

Sub-Index	**01$_h$**			
Name	**min_position_range_limit**			
Info	position_unit	rw	PDO	INT32
Value	--	--		

Sub-Index	**02$_h$**			
Name	**max_position_range_limit**			
Info	position_unit	rw	PDO	INT32
Value	--	--		

3.7.2.16 Object 6510_h_20_h: position_range_limit_enable

Via the object position_range_limit_enable the range limits defined by the object $607B_h$ can be activated. Different modes are possible:

If the mode "Shortest distance" is selected, positioning is always carried out on the physically shorter distance to the target. The drive itself adjusts the sign of the travel speed for this purpose. In the two modes with fixed direction of rotation, positioning is always carried out only in the direction specified in the mode.

Index	**6510_h**			
Name	**drive_data**			
Type	RECORD			$F0_h$

Sub-Index	**20_h**			
Name	**position_range_limit_enable**			
Info	--	rw	~~PDO~~	UINT16
Value	0...5	--		

Value	Description
0	Off
1	Shortest distance (for compatibility reasons)
2	Shortest distance
3	Reserved
4	Direction always „positive“
5	Direction always „negative“

3.8 Setpoint limitation

3.8.1 Object 2415$_h$: current_limitation

The current_limitation object record can be used to limit the maximum current for the motor in the Profile Position Mode, Interpolated Position Mode, Cyclic Synchronous Position Mode, Homing Mode and Profile Velocity Mode, thus allowing torque-limited speed operation, for example. The limit_current_input_channel object is used to specify the source of the limiting torque setpoint. Here you can choose between setting a direct setpoint (fixed value) or using an analogue input. The limit_current object is used to specify either the limiting torque (source = fixed value) or the scaling factor for the analogue inputs (source = AINx), depending on the selected source. In the first case, the current is directly limited to the torque-proportional fixed value in mA, in the second case the current in mA is specified, which should correspond to an applied voltage of 10V.

Index	**2415$_h$**			
Name	**current_limitation**			
Type	RECORD			02$_h$

Sub-Index	**01$_h$**			
Name	**limit_current_input_channel**			
Info	--	rw	PDO	INT8
Value	0...4	0		

Sub-Index	**02$_h$**			
Name	**limit_current**			
Info	mA	rw	PDO	INT32
Value	--	--		

Value	Description
0	No limit
1	AIN0
2	AIN1
3	AIN2
4	Fixed value / fieldbus (fieldbus selector 2)

3.8.2 Object 2416_h: speed_limitation

The speed_limitation object group can be used to limit the maximum speed of the motor in Profile Torque Mode, thus allowing speed-limited torque operation.
The limit_speed_input_channel object is used to specify the setpoint source of the limiting speed. Here you can choose between setting a direct setpoint (fixed value) or using an analogue input. The limit_speed object is used to specify either the limiting speed (source = fixed value) or the scaling factor for the analog inputs (source = AINx), depending on the selected source. In the first case, the speed is directly limited to the fixed value, in the second case the speed is specified, which should correspond to an applied voltage of 10V.

Index	**2416_h**			
Name	**speed_limitation**			
Type	RECORD			02_h

Sub-Index	**01_h**			
Name	**limit_speed_input_channel**			
Info	--	rw	~~PDO~~	INT8
Value	0...4	0		

Sub-Index	**02_h**			
Name	**limit_speed**			
Info	speed_unit	rw	~~PDO~~	INT32
Value	--	--		

Value	Description
0	No limit
1	AIN0
2	AIN1
3	AIN2
4	Fixed value / fieldbus (fieldbus selector 2)

3.9 Encoder adaptation

3.9.1 Overview

This chapter describes the configuration of the angle encoder input X2A, X2B and the master frequency input (BL 4000-C, BL 4000-M / BL 4000-D: X1).

NOTICE **Damage to property due to incorrect angle encoder settings**

Incorrect angle encoder settings can cause the drive to rotate uncontrollably and possibly destroy parts of the machine.

3.9.2 Description of objects

3.9.2.1 Object 2024_h: encoder_x2a_data_field

The object record encoder_x2a_data_field contains parameters that are necessary for the operation of the angle encoder at connector X2A.

Since many encoder settings are only effective after a reset, the selection and setting of the encoders should be done via the MPC ServoCommander. The following settings can be read or changed via CANopen:

The object encoder_x2a_resolution specifies how many increments are generated by the encoder per revolution or length unit. Since only resolvers can be connected to input X2A, which are always evaluated with 16 bits, 65536 is always returned here. The objects encoder_x2a_numerator and encoder_x2a_divisor can be used to take into account a possible gear (also with sign) between motor shaft and encoder.

Index	**2024_h**			
Name	**encoder_x2a_data_field**			
Type	RECORD			03_h
Sub-Index	**01_h**			
Name	**encoder_x2a_resolution**			
Info	Increments (4 * line count)	ro	~~PDO~~	UINT32
Value	--	--		
Sub-Index	**02_h**			
Name	**encoder_x2a_numerator**			
Info	--	rw	~~PDO~~	INT16
Value	-32768...-1,1...32767	1		
Sub-Index	**03_h**			
Name	**encoder_x2a_divisor**			
Info	--	rw	~~PDO~~	INT16
Value	1...32767	1		

3.9.2.2 Object 2026_h: encoder_x2b_data_field

The object record encoder_x2b_data_field contains parameters that are necessary for the operation of the angle encoder at connector X2B.

The object encoder_x2b_resolution specifies how many increments are generated by the encoder per revolution (for incremental encoders this is four times the number of lines or periods per revolution) or length unit. The object encoder_x2b_counter returns the currently counted number of increments, i.e. values between 0 and encoder_x2b_resolution-1.

The objects encoder_x2b_numerator and encoder_x2b_divisor can be used to take into account a possible gear (also with sign) between motor shaft and encoder.

Index	**2026_h**	
Name	**encoder_x2b_data_field**	
Type	RECORD	16_h

Sub-Index	**01_h**			
Name	**encoder_x2b_resolution**			
Info	Increments (4 * line count)	rw	~~PDO~~	UINT32
Value	--	--		

Sub-Index	**02_h**			
Name	**encoder_x2b_numerator**			
Info	--	rw	~~PDO~~	INT16
Value	-32768...-1,1...32767	1		

Sub-Index	**03_h**			
Name	**encoder_x2b_divisor**			
Info	--	rw	~~PDO~~	INT16
Value	1...32767	1		

Sub-Index	**04_h**			
Name	**encoder_x2b_counter**			
Info	Increments (4 * line count)	ro	PDO	UINT32
Value	0 … (encoder_x2b_resolution – 1)	--		

3.9.2.3 Object 2025_h: encoder_x10_data_field

The object record encoder_x10_data_field contains parameters that are necessary for the operation of the master frequency input, which is located on connector X1 in the BL 4000 devices series.

A digital incremental encoder or emulated incremental signals, for example from another servo drive (master frequency output), can be connected to the master frequency input. The signals of the master frequency input can optionally be used as setpoint or actual value.

The object encoder_x10_resolution specifies how many increments are generated by the encoder per revolution (for incremental encoders this is four times the number of lines or periods per revolution) or length unit. The object encoder_x10_counter returns the currently counted number of increments, i.e. values between 0 and encoder_x10_resolution-1.

The objects encoder_x10_numerator and encoder_x10_divisor can be used to take into account a possible gear (also with sign) between motor shaft and encoder. When using the master frequency input as setpoint, this can be used to realise gear ratios between master and slave.

Index	**2025_h**			
Name	**encoder_x10_data_field**			
Type	RECORD			05_h
Sub-Index	**01_h**			
Name	**encoder_x10_resolution**			
Info	Increments (4 * line count)	rw	~~PDO~~	UINT32
Value	encoder dependent	--		
Sub-Index	**02_h**			
Name	**encoder_x10_numerator**			
Info	--	rw	~~PDO~~	INT16
Value	-32768...-1,1...32767	1		
Sub-Index	**03_h**			
Name	**encoder_x10_divisor**			
Info	--	rw	~~PDO~~	INT16
Value	1...32767	1		
Sub-Index	**04_h**			
Name	**encoder_x10_counter**			
Info	Increments (4 * line count)	ro	PDO	UINT32
Value	0 … (encoder_x10_resolution – 1)	--		
Sub-Index	**05_h**			
Name	**encoder_x10_position**			
Info	--	ro	PDO	INT32
Value	--	--		

3.9.2.4 Object $202C_h$: max_comm_enc_pos_enc_difference

The object max_comm_enc_pos_enc_difference returns the maximum difference between the commutation encoder and the actual position encoder.

Index	**$202C_h$**			
Name	**max_comm_enc_pos_enc_difference**			
Info	position_unit	rw	PDO	INT32
Value	--	--		

3.10 Master frequency output

3.10.1 Overview

This object group is used to parameterise the master frequency output BL 4000-C, BL 4000-M / BL 4000-D: X1). Thus, master-slave applications in which the master frequency output (incremental encoder emulation) of the master is connected to the master frequency input of the slave can be parameterised via CANopen.

3.10.2 Description of objects

3.10.2.1 Object $201A_h$: encoder_emulation_data

The object record encoder_emulation_data contains all options for the master frequency output.

Using the object encoder_emulation_resolution the output number of increments (= four times the number of lines) can be set as a multiple of 4. In a master-slave application, this must correspond to the encoder_X10_resolution of the slave to achieve a ratio of 1:1.

With the object encoder_emulation_offset the position of the output zero pulse can be shifted in relation to the zero position of the actual value encoder.

Index	**$201A_h$**	
Name	**encoder_emulation_data**	
Type	RECORD	02_h

Sub-Index	**01_h**			
Name	**encoder_emulation_resolution**			
Info	Increments (4 * line count)	rw	~~PDO~~	INT32
Value	4 * (1…8192)	--		

Sub-Index	**02_h**			
Name	**encoder_emulation_offset**			
Info	32767 = 180°	rw	~~PDO~~	INT16
Value	-32768…32767	--		

3.10.2.2 Object 2028_h: encoder_emulation_resolution

The object encoder_emulation_resolution only exists for compatibility reasons. It corresponds to object $201A_h\_01_h$.

3.11 Setpoint / actual value selection

3.11.1 Overview

The following objects can be used to change the source for the setpoint and the actual value. By default, the servo drive uses the input for the motor encoder X2A or X2B as the actual value for the position controller. When using an external position encoder, e.g. behind a gearbox, the position value fed in via the master frequency input can be used as the actual value for the position controller. Furthermore, it is possible to use the master frequency input as an additional setpoint, which allows synchronous operating modes. For reasons of downward compatibility, the objects for parameterising the master frequency input are always designated "_X10_", even if the master frequency input is located on the connector [X1], as is the case with the BL 4000 controller family.

3.11.2 Description of objects

3.11.2.1 Object 201F$_h$: commutation_encoder_select

The object commutation_encoder_select specifies the encoder input that is used as commutation encoder. Since this value only becomes effective after a reset, the commutation encoder should always be set via the MPC ServoCommander .

Index	**201F$_h$**			
Name	**commutation_encoder_select**			
Info	--	rw	~~PDO~~	INT16
Value	0, 2	--		

Value	Description
0	X2A
2	X2B

3.11.2.2 Object 2021$_h$: position_encoder_selection

The object position_encoder_selection specifies the encoder input that is used to determine the actual position (*actual position encoder*). This value can be changed in order to switch to "position control via an external encoder" (connected to the driven side). It is possible to switch between master frequency input and the encoder input that is selected as *commutation encoder* (X2A or X2B). If one of the encoder inputs X2A / X2B is selected as actual position encoder, the one used as *commutation encoder* must be used. If the respective other encoder is selected, the system automatically switches over to the *commutation encoder*.

Index	**2021$_h$**			
Name	**position_encoder_selection**			
Info	--	rw	~~PDO~~	INT16
Value	0...2	--		

Value	Description
0	X2A
1	X2B
2	Master frequency input

INFORMATION **Permissible combinations**

The following combinations are **permitted**:

Commutating encoder X2A, position encoder: master frequency input
Commutating encoder X2B, position encoder: master frequency input

The following combinations are **not permitted**:

Commutating encoder X2A, position encoder: X2B
Commutating encoder X2B, position encoder: X2A

3.11.2.3 Object 2022$_h$: synchronisation_encoder_selection

The object synchronisation_encoder_selection specifies the encoder input via which the synchronisation setpoint is fed in. Depending on the operating mode, this is equivalent to a position setpoint (Profile Position Mode) or a speed setpoint (Profile Velocity Mode).

Only the master frequency input can be used as synchronisation input. Thus, it is possible to select between "Master frequency input" and "No encoder". Do not select the same input as used for the actual value encoder as the synchronisation setpoint.

Index	**2022$_h$**			
Name	**synchronisation_encoder_selection**			
Info	--	rw	~~PDO~~	INT16
Value	-1, 2	--		

Value	Description
-1	No encoder / undefined
2	Master frequency input

3.11.2.4 Object 202F$_h$: synchronisation_selector_data

The object synchronisation_main can be used to activate a synchronous setpoint. Bit 0 must be set so that the synchronous setpoint is calculated at all. Bit 1 enables the synchronous position to be switched on only after starting a position set (flying saw). Bit 8 can be used to specify that the homing run should be executed without switching on the synchronous position in order to be able to reference the master and slave separately.

Index	**202F$_h$**	
Name	**synchronisation_selector_data**	
Type	RECORD	07$_h$

Sub-Index	**07$_h$**			
Name	**synchronisation_main**			
Info	--	rw	~~PDO~~	UINT16
Value	see Table	--		

Bit	Value	Description
0	0001$_h$	0: Synchronisation inactive 1: Synchronisation active
1	0002$_h$	0: "Flying saw" inactive 1: "Flying saw" active
8	0100$_h$	0: Synchronization during homing 1: No synchronization during homing

3.11.2.5 Object 2023$_h$: synchronisation_filter_time

The object synchronisation_filter_time is used to define the filter time constant of a PT1 filter with which the synchronisation speed is smoothed. This may be necessary especially with low line numbers, since even small changes of the input value correspond to high speeds. On the other hand, the drive may no longer be able to follow a dynamic input signal fast enough at high filter times.

Index	**2023$_h$**			
Name	**synchronisation_filter_time**			
Info	µs	rw	~~PDO~~	UINT32
Value	10...50000	--		

3.12 Analogue inputs

3.12.1 Overview

The servo drives have analogue inputs, which may be used to provide setpoints to the servo drive, for example. For all these analogue inputs, the following objects offer the possibility of reading out the current input voltage (analog_input_voltage) and setting an offset (analog_input_offset). Depending on the servo drive series (BL 4000-M / BL 4000-D, BL 4000-C), there are different numbers of analogue inputs.

3.12.2 Description of objects

3.12.2.1 Object 2400_h: analog_input_voltage

The object group analog_input_voltage supplies the current input voltage of the respective channel in millivolts including the offset.

Index	**2400_h**			
Name	**analog_input_voltage**			
Type	ARRAY			03_h

Sub-Index	**01_h**			
Name	**analog_input_voltage_ch_0**			
Info	mV	ro	PDO	INT16
Value	--	--		

Sub-Index	**02_h**			
Name	**analog_input_voltage_ch_1**			
Info	mV	ro	PDO	INT16
Value	--	--		

Sub-Index	**03_h**			
Name	**analog_input_voltage_ch_2**			
Info	mV	ro	PDO	INT16
Value	--	--		

3.12.2.2 Object 2401_h: analog_input_offset

Via the object group analog_input_offset the offset voltage in millivolts can be set or read for the respective inputs. With the help of the offset, a possible applied DC voltage can be compensated. A positive offset compensates a positive input voltage.

Index	**2401_h**			
Name	**analog_input_offset**			
Type	ARRAY			03_h

Sub-Index	**01_h**			
Name	**analog_input_offset_ch_0**			
Info	mV	rw	~~PDO~~	INT32
Value	-10000...10000	--		

Sub-Index	**02_h**			
Name	**analog_input_offset_ch_1**			
Info	mV	rw	~~PDO~~	INT32
Value	-10000...10000	--		

Sub-Index	**03_h**			
Name	**analog_input_offset_ch_2**			
Info	mV	rw	~~PDO~~	INT32
Value	-10000...10000	--		

3.13 Digital inputs and outputs

3.13.1 Overview

All digital inputs of the servo drive can be read via the CAN bus and almost all digital outputs can be set as required. Furthermore, status messages can be assigned to the digital outputs of the servo drive. Depending on the devices series, not all digital inputs/outputs described here may be available for every device.

3.13.2 Description of objects

3.13.2.1 Object $60FD_h$: digital_inputs

Via the object $60FD_h$ the digital inputs may be read:

Index	**$60FD_h$**			
Name	**digital_inputs**			
Info	--	ro	PDO	UINT32
Value	see Table	--		

Bit	Value	Digital input
0	00000001_h	Negative limit switch
1	00000002_h	Positive limit switch
2	00000004_h	Reference switch
3	00000008_h	Interlock (Controller enable or Powerstage enable or STO missing)
24...27	$0F000000_h$	DIN0...DIN3
28	10000000_h	DIN8
29	20000000_h	BL 4100-C: DIN4

3.13.2.2 Object $60FE_h$: digital_outputs

Via object $60FE_h$ the digital outputs may be controlled. A set bit in object digital_outputs_mask specifies which digital output is to be controlled. Via the digital_outputs_data object the selected outputs can then be set as required. Please note that a delay of up to 10 ms can occur when controlling the digital outputs. When the outputs are actually set can be determined by reading back object $60FE_h$.

Index	**60FE$_h$**			
Name	**digital_outputs**			
Type	ARRAY			02$_h$

Sub-Index	**01$_h$**			
Name	**digital_outputs_data**			
Info	--	rw	PDO	UINT32
Value	--	--		

Sub-Index	**02$_h$**			
Name	**digital_outputs_mask**			
Info	--	rw	PDO	UINT32
Value	--	--		

Bit	Value	Digital output
0	00000001$_h$	1 = Applying the brake
25...27	0E000000$_h$	DOUT1...DOUT3

NOTICE **Damage to property possible**

If control of the brake is enabled via digital_output_mask, the holding brake is released manually by clearing bit 0 in digital_output_data!

This can cause the axis to drop in the case of hanging axes.

3.13.2.3 Object 2420$_h$: digital_output_state_mapping

The object group digital_outputs_state_mapping can be used to issue various status messages of the servo drive via the digital outputs. For the integrated digital outputs of the servo drive, there is a separate sub-index for each output.
If such a status message has been assigned to a digital output and the output is then switched on or off directly via digital_outputs (60FE$_h$), the digital_outputs_state_mapping object is also set to OFF (0) or ON (12).

Index	**2420$_h$**			
Name	**digital_outputs_state_mapping**			
Type	RECORD			12$_h$

Sub-Index	**01$_h$**			
Name	**dig_out_state_mapp_dout_1**			
Info	--	rw	~~PDO~~	UINT8
Value	0...16, see Table	--		

Sub-Index	**02$_h$**			
Name	**dig_out_state_mapp_dout_2**			
Info	--	rw	~~PDO~~	UINT8
Value	0...16, see Table	--		

Sub-Index	**03_h**			
Name	**dig_out_state_mapp_dout_3**			
Info	--	rw	~~PDO~~	UINT8
Value	0...16, see Table	--		

Value	Description	Value	Description
0	Off (Output is low)	9	Undervoltage intermed. circuit
1	Position $X_{set} = X_{dest}$	10	Brake unlocked
2	Position $X_{act} = X_{dest}$	11	Power stage active
3	Reserved	12	On (Output is high)
4	Remaining distance	13	Reserved
5	Homing active	14	Reserved
6	Speed reached	15	Linear motor identified
7	I²t monitoring active	16	Homing position valid
8	Following error		

3.14 Limit switch / Reference switch

3.14.1 Overview

For defining the reference position of the servo drive, either limit switches or homing switches can be used. More information about the possible homing methods can be found in section 5.2 *Homing Mode* on page 124.

3.14.2 Description of objects

3.14.2.1 Object 6510_h_11_h: limit_switch_polarity

The polarity of the limit switches can be programmed by the object 6510_h_11_h (limit_switch_polarity). A zero must be entered in this object for normally closed contacts, a one must be entered when using normally open contacts.

Index	**6510_h**	
Name	**drive_data**	
Type	RECORD	$F0_h$

Sub-Index	**11_h**			
Name	**limit_switch_polarity**			
Info	--	rw	~~PDO~~	INT16
Value	0, 1	1		

Value	Description
0	Normally closed contact
1	Normally open contact

3.14.2.2 Object $6510_h\_12_h$: limit_switch_selector

Via object $6510_h\_12_h$ (limit_switch_selector) the assignment of the limit switches (negative, positive) can be swapped without having to make changes to the cabling. To exchange the assignment of the limit switches, enter a one.

Index	**6510_h**			
Name	**drive_data**			
Type	RECORD			$F0_h$

Sub-Index	**12_h**			
Name	**limit_switch_selector**			
Info	--	rw	~~PDO~~	INT16
Value	0, 1	0		

Value	Description
0	DIN6 = E0 (negative limit switch) DIN7 = E1 (positive limit switch)
1	DIN6 = E1 (positive limit switch) DIN7 = E0 (negative limit switch)

3.14.2.3 Object $6510_h\_15_h$: limit_switch_deceleration

The limit_switch_deceleration object determines the deceleration used for braking when the limit switch is reached during normal operation (limit switch emergency ramp).

Index	**6510_h**			
Name	**drive_data**			
Type	RECORD			$F0_h$

Sub-Index	**15_h**			
Name	**limit_switch_deceleration**			
Info	acceleration_unit	rw	~~PDO~~	INT32
Value	0...3000000 min^{-1}/s	--		

3.14.2.4 Object $6510_h\_14_h$: homing_switch_polarity

The polarity of the reference switch can be configured by object $6510_h\_14_h$ (homing_switch_polarity). For an opening reference switch, a zero must be entered in this object, for the use of closing contacts a one must be entered.

Index	**6510_h**			
Name	**drive_data**			
Type	RECORD			$F0_h$

Sub-Index	**14_h**			
Name	**homing_switch_polarity**			
Info	--	rw	~~PDO~~	INT16
Value	0, 1	1		

Value	Description
0	Normally closed contact
1	Normally open contact

3.14.2.5 Object $6510_h\_13_h$: homing_switch_selector

Object $6510_h\_13_h$ (homing_switch_selector) determines whether DIN8 or DIN9 should be used as input for the reference switch.

Index	**6510_h**			
Name	**drive_data**			
Type	RECORD			$F0_h$

Sub-Index	**13_h**			
Name	**homing_switch_selector**			
Info	--	rw	~~PDO~~	INT16
Value	0, 1	0		

Value	Description
0	DIN9
1	DIN8

3.15 Position capturing (Sampling)

3.15.1 Overview

The servo drives offer the possibility of capturing the actual position value on the rising or falling edge of a digital input (e.g. a measuring probe). This position value can then be read out, e.g. for calculation within a control system.

All necessary objects are summarised in the record sample_data: The object sample_mode determines the type of sampling: Should only a single sample event be recorded or should sampling be continuous. Using the object sample_status, the controller can query whether a sample event has occurred. This is indicated by a set bit, which can also be displayed in the statusword if the object sample_status_mask is set accordingly.

The object sample_control is used to control the release of the sample event and the sampled positions can be read out via the objects sample_position_rising_edge and sample_position_falling_edge.

Which digital input is used can be defined with the MPC ServoCommander under Parameters / IOs / Digital Inputs / Sample Input.

3.15.2 Description of objects

3.15.2.1 Object 204A$_h$: sample_data

Index	**204A$_h$**	
Name	**sample_data**	
Type	RECORD	06$_h$

The following object can be used to select whether the position is to be determined on each occurrence of a sample event (continuous sampling) or whether sampling is to be disabled after a sample event until sampling is enabled again. Please note that even a bouncing input can trigger both edges.

Sub-Index	**01$_h$**			
Name	**sample_mode**			
Info	--	rw	~~PDO~~	UINT16
Value	0...1	--		

Value	Description
0	Continuous sampling
1	Autolock sampling

The following object indicates a new sample event.

Sub-Index	**02_h**			
Name	**sample_status**			
Info	--	ro	PDO	UINT8
Value	0...3	--		

Bit	Value	Name	Description
0	01_h	`falling_edge_occurred`	= 1: Position sampled (falling edge)
1	02_h	`rising_edge_occurred`	= 1: Position sampled (rising edge)

The following object can be used to specify those bits of the sample_status object that should also lead to the setting of bit 15 of the statusword. This means that the information "Sample event occurred" is available in the statusword, which is usually transferred anyway. Only if "Sample event occurred" is displayed there, the controller must read the sample_status object to determine which edge has occurred.

Sub-Index	**03_h**			
Name	**sample_status_mask**			
Info	--	rw	PDO	UINT8
Value	0...3	--		

Bit	Value	Name	Description
0	01_h	`falling_edge_visible`	If `falling_edge_occured` = 1 Bit 15 is set in the statusword
1	02_h	`rising_edge_visible`	If `rising_edge_occured` = 1 Bit 15 is set in the statusword

Setting the respective bit in sample_control resets the corresponding status bit in sample_status and, in case of "Autolock" sampling, enables sampling again.

Sub-Index	**04_h**			
Name	**sample_control**			
Info	--	wo	PDO	UINT8
Value	0...3	0		

Bit	Value	Name	Description
0	01_h	`falling_edge_enable`	Sampling on falling edge
1	02_h	`rising_edge_enable`	Sampling on rising edge

The following objects contain the sampled positions.

Sub-Index	**05_h**			
Name	**sample_position_rising_edge**			
Info	position_unit	ro	PDO	INT32
Value	--	--		

Sub-Index	**06_h**			
Name	**sample_position_falling_edge**			
Info	position_unit	ro	PDO	INT32
Value	--	--		

3.16 Brake control

3.16.1 Overview

The following objects can be used to parameterise how the servo drive controls a holding brake that may be integrated in the motor. The holding brake is always enabled as soon as the servo drive enable is switched on. For holding brakes with high mechanical inertia, a delay time t_A can be parameterised so that the holding brake is engaged before the power stage is switched off (sagging of vertical axes). Similarly, the control of the motor is delayed (t_F) until the holding brake is completely released. Both delays are parameterised simultaneously by the object brake_delay_time ($t_A = t_F$).

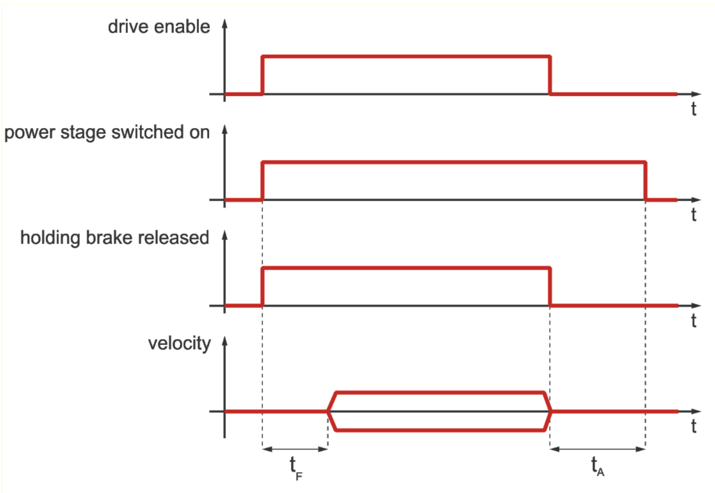


Figure 8: Function of brake delay (for speed control / positioning)

3.16.2 Description of objects

3.16.2.1 Object 6510_h_18_h: brake_delay_time

The braking delay time can be parameterised via the object brake_delay_time.

Index	**6510_h**	
Name	**drive_data**	
Type	RECORD	$F0_h$

Sub-Index	**18_h**			
Name	**brake_delay_time**			
Info	ms	rw	~~PDO~~	UINT16
Value	0...32000	--		

3.17 Device information

Numerous CAN objects can be used to read a wide variety of information from the device, such as servo drive type, firmware used, etc.

3.17.1 Description of objects

3.17.1.1 Object 1000_h: device_type

The device_type object indicates in the lower 16 bits that device profile 402 is supported and in the upper 16 bits that it is a servo drive (bit 17).

Index	**1000_h**			
Name	**device_type**			
Info	--	ro	~~PDO~~	UINT32
Value	--	00020192_h		

3.17.1.2 Object 1008_h: manufacturer_device_name

Via the object manufacturer_device_name the name of the device series can be read in plain text.

Index	**1008_h**			
Name	**manufacturer_device_name**			
Info	--	ro	~~PDO~~	VISSTR
Value	--	--		

3.17.1.3 Object 1009_h: manufacturer_hardware_version

The manufacturer_hardware_version object can be used to read the hardware revision of the device. This is also displayed in the MPC ServoCommander under Help / Info Tab Firmware / Hardware.

Index	**1009_h**			
Name	**manufacturer_hardware_version**			
Info	MMM.SSS	ro	~~PDO~~	VISSTR
Value	--	--		

Value	Description
M	main version
S	sub version

3.17.1.4 Object 100A$_h$: manufacturer_software_version

The manufacturer_software_version object can be used to read the firmware version in plain text. The individual parts of the version number are formatted as ASCII characters without leading zeros and are separated by dots, e.g. "1.0.0.1.2".

Index	**100A$_h$**			
Name	**manufacturer_software_version**			
Info	M.S.C.K.k	ro	~~PDO~~	VISSTR
Value	--	--		

Value	Description
M	Corresponds to MMMM of *Object 6510h_A9h: firmware_main_version*
S	Corresponds to SSSS of *Object 6510h_A9h: firmware_main_version*
C	Corresponds to *Object 6510h_AAh: firmware_custom_version*
K	Corresponds to MMMM of *Object 6510h_ADh: km_release*
k	Corresponds to SSSS of *Object 6510h_ADh: km_release*

3.17.1.5 Object 1018$_h$: identity_object

The servo drive can be uniquely identified in a CANopen network via the identity_object defined in DS301. For this purpose, the vendor code (vendor_id), a unique product code (product_code), the revision number of the CANopen implementation (revision_number) and the serial number (serial_number) can be read out.

Index	**1018$_h$**	
Name	**identity_object**	
Type	RECORD	04$_h$

Sub-Index	**01$_h$**			
Name	**vendor_id**			
Info	--	ro	~~PDO~~	UINT32
Value	--	000000E4$_h$		

Sub-Index	**02$_h$**			
Name	**product_code**			
Info	--	ro	~~PDO~~	UINT32
Value	--	--		

Value	Description
8202$_h$	BL 4102-C
8203$_h$	BL 4104-C
8208$_h$	BL 4304-C
8209$_h$	BL 4308-C

Value	Description
8212_h	BL 4312-C
$820A_h$	BL 4104-M ETH
$820C_h$	BL 4104-D ETH
$820D_h$	BL 4840-M ETH
$820F_h$	BL 4840-D ETH
$820B_h$	BL 4104-M CAN
8210_h	BL 4104-D CAN
$820E_h$	BL 4840-M CAN
8211_h	BL 4840-D CAN

Sub-Index	**03_h**			
Name	**revision_number**			
Info	--	ro	~~PDO~~	UINT32
Value	--	00040002_h		

Sub-Index	**04_h**			
Name	**serial_number**			
Info	--	ro	~~PDO~~	UINT32
Value	--	--		

3.17.1.6 Object 6510_h_$A0_h$: drive_serial_number

The object drive_serial_number returns the serial number of the servo drive . This object is used to ensure compatibility with earlier versions.

Index	**6510_h**	
Name	**drive_data**	
Type	RECORD	$F0_h$

Sub-Index	**$A0_h$**			
Name	**drive_serial_number**			
Info	--	ro	~~PDO~~	UINT32
Value	--	--		

3.17.1.7 Object 6510_h_$A1_h$: drive_type

The drive_type object can be used to read the product code of the servo drive. This object is used to ensure compatibility with earlier versions.

Index	**6510_h**	
Name	**drive_data**	
Type	RECORD	$F0_h$

Sub-Index	**$A1_h$**			
Name	**drive_type**			
Info	see 1018_h_02_h (product code)	ro	~~PDO~~	UINT32
Value	see 1018_h_02_h (product code)	--		

3.17.1.8 Object 6510_h_$A9_h$: firmware_main_version

The firmware_main_version object can be used to read the main version number of the firmware (product step).

Index	**6510_h**	
Name	**drive_data**	
Type	RECORD	$F0_h$

Sub-Index	**$A9_h$**			
Name	**firmware_main_version**			
Info	$MMMMSSSS_h$	ro	~~PDO~~	UINT32
Value	--	--		

Value	Description
M	main version
S	sub version

3.17.1.9 Object 6510_h_AA_h: firmware_custom_version

The object firmware_custom_version can be used to read the version number of the customer-specific variant of the firmware.

Index	**6510_h**	
Name	**drive_data**	
Type	RECORD	$F0_h$

Sub-Index	**AA_h**			
Name	**firmware_custom_version**			
Info	--	ro	~~PDO~~	UINT32
Value	--	--		

3.17.1.10 Object $6510_h\_AD_h$: km_release

The version number of the km_release can be used to differentiate between firmware versions of the same product level.

Index	**6510_h**			
Name	**drive_data**			
Type	RECORD			$F0_h$

Sub-Index	**AD_h**			
Name	**km_release**			
Info	$MMMMSSSS_h$	ro	~~PDO~~	UINT32
Value	--	--		

Value	Description
M	main version
S	sub version

3.17.1.11 Object $6510_h\_AC_h$: firmware_type

The firmware_type object is present for compatibility reasons and has no significance for the BL 4000 series.

Index	**6510_h**			
Name	**drive_data**			
Type	RECORD			$F0_h$

Sub-Index	**AC_h**			
Name	**firmware_type**			
Info	$000000GX_h$	ro	~~PDO~~	UINT32
Value	$F2_h$	--		

3.17.1.12 Object $6510_h\_B0_h$: cycletime_current_controller

The object cycletime_current_controller returns the cycle time of the current controller in microseconds.

Index	**6510_h**			
Name	**drive_data**			
Type	RECORD			$F0_h$

Sub-Index	**$B0_h$**			
Name	**cycletime_current_controller**			
Info	µs	ro	~~PDO~~	UINT32
Value	--	--		

3.17.1.13 Object 6510_h_$B1_h$: cycletime_velocity_controller

The object cycletime_velocity_controller returns the cycle time of the speed controller in microseconds.

Index	**6510_h**	
Name	**drive_data**	
Type	RECORD	$F0_h$

Sub-Index	**$B1_h$**			
Name	**cycletime_velocity_controller**			
Info	µs	ro	~~PDO~~	UINT32
Value	--	--		

3.17.1.14 Object 6510_h_$B2_h$: cycletime_position_controller

The object cycletime_position_controller returns the cycle time of the position controller in microseconds.

Index	**6510_h**	
Name	**drive_data**	
Type	RECORD	$F0_h$

Sub-Index	**$B2_h$**			
Name	**cycletime_position_controller**			
Info	µs	ro	~~PDO~~	UINT32
Value	--	--		

3.17.1.15 Object 6510_h_$B3_h$: cycletime_trajectory_generator

The object cycletime_trajectory_generator returns the cycle time of the trajectory generator in microseconds.

Index	**6510_h**	
Name	**drive_data**	
Type	RECORD	$F0_h$

Sub-Index	**$B3_h$**			
Name	**cycletime_trajectory_generator**			
Info	µs	ro	~~PDO~~	UINT32
Value	--	--		

3.17.1.16 Object 6510$_h$_C0$_h$: commissioning_state

NOTICE **Unsuitable parameterisation possible**

This object does not contain any information about whether the servo drive has been parameterised correctly according to the motor and the application, but only whether the points mentioned were parameterised at least once after delivery.

INFORMATION **„A“ on the 7-segment display**

Note that at least one bit must be set in the commissioning_state object to suppress the "A" on the display of your servo drive.

Index	**6510$_h$**			
Name	**drive_data**			
Type	RECORD			F0$_h$
Sub-Index	**C0$_h$**			
Name	**commissioning_state**			
Info	--	rw	~~PDO~~	UINT32
Value	--	--		

Bit	Description	Bit	Description
0	Nominal current valid	9	Reserved
1	Maximum current valid	10	Physical units valid
2	Number of poles of motor valid	11	Speed controller valid
3	Offset angle / direction of rotation valid	12	Position controller valid
4	Reserved	13	Monitoring parameter valid
5	Offset angle / direction of rotation Hall sensor valid	14	Reserved
6	Reserved	15	Limit switch polarity valid
7	Absolute position encoder system valid	16...31	Reserved
8	Current controller parameters valid		

3.17.1.17 Object 20FD$_h$: user_device_name

The user_device_name object can be used to read and write the user-definable name of the drive (e.g. "X-axis").

Index	**20FD$_h$**			
Name	**user_device_name**			
Info	--	rw	~~PDO~~	VISSTR
Value	--	--		

3.18 Error management

3.18.1 Overview

The servo drives offer the possibility to change the error reaction of individual events, such as the occurrence of a following error. As a result, the servo drive reacts differently when a particular event occurs: Depending on the setting, the servo drive will decelerate, the power stage will be switched off immediately or only a warning will be shown on the display.

A fixed minimum reaction is provided for each event, which must not be fallen below. This means that "critical" errors such as "06-0 short-circuit of the power stage" cannot be reparameterised, as in this case an immediate switch-off is necessary to protect the servo drive from being destroyed.

If a lower error response than permitted for the respective error is entered, the value is limited to the lowest permitted error response. A list of all error numbers can be found in the “Product Manual smartServo BL 4000”.

3.18.2 Description of objects

3.18.2.1 Object 2100_h: error_management

Index	**2100_h**	
Name	**error_management**	
Type	RECORD	02_h

In the object error_number the main error number is specified whose reaction should be changed. The main error number is the one usually given before the hyphen (for example, error 08-2, main error number 8).

Sub-Index	**01_h**			
Name	**error_number**			
Info	--	rw	~~PDO~~	UINT8
Value	1...96	--		

The reaction of the error can be changed in the object error_reaction_code. If the response is less than the manufacturer's minimum response, the system limits the error to this. The actual reaction set can be determined by reading it back.

Sub-Index	**02_h**			
Name	**error_reaction_code**			
Info	--	rw	~~PDO~~	UINT8
Value	0, 1, 3, 5, 7, 8	--		

Value	Description
0	No action
1	Entry into buffer
3	Warning on the 7-segment display
5	Disable servo drive
7	Stop at maximum current
8	Disable power stage immediately

3.18.2.2 Object $200F_h$: last_warning_code

Warnings are remarkable events of the drive (e.g. a following error), which, in contrast to an error, should not lead to a shutdown of the drive. Warnings are shown on the 7-segment display of the servo drive and are then automatically reset.

The last occurred warning can be read out via the following object: Thereby bit 15 indicates whether the warning is currently still active.

Index	**$200F_h$**			
Name	**last_warning_code**			
Info	--	ro	PDO	UINT16
Value	--	--		

Bit	Value	Description
0... 3	$000F_h$	Warning sub-number
4...11	$0FF0_h$	Warning main number
15	8000_h	Warning is active

4 Device Control

4.1 Overview

The following chapter describes how the servo drive is controlled under CANopen, i.e. how the power stage is switched on or an error is acknowledged.

Under CANopen, the entire control of the servo drive can be realised via two objects: The host can operate the servo drive via the controlword, while the status of the servo drive can be read back in the statusword. The following terms are used to explain servo drive control:

Keyword	Explanation
State	The servocontroller is in different states depending on whether the power stage is switched on or an error has occurred. The states defined under CANopen are presented in the following chapter. Example: `OPERATION_ENABLE`
State Transition	Like the states, CANopen also defines how to go from one state to another (e.g. to acknowledge an error). State transitions are triggered by the host by setting bits in the controlword or internally by the servocontroller if it detects an error, for example
Command	To trigger state transitions, certain combinations of bits must be set in the controlword. Such a combination is called a command. Example: Enable Operation
State Machine	The states and state transitions together form the State Machine diagram, i.e. the overview of all states and possible transitions.

4.2 State Machine

The status diagram can be roughly divided into three areas: "Power Disabled" means that the power stage is switched off and "Power Enabled" means that the power stage is switched on. The "Fault" area summarises the states necessary for error handling.

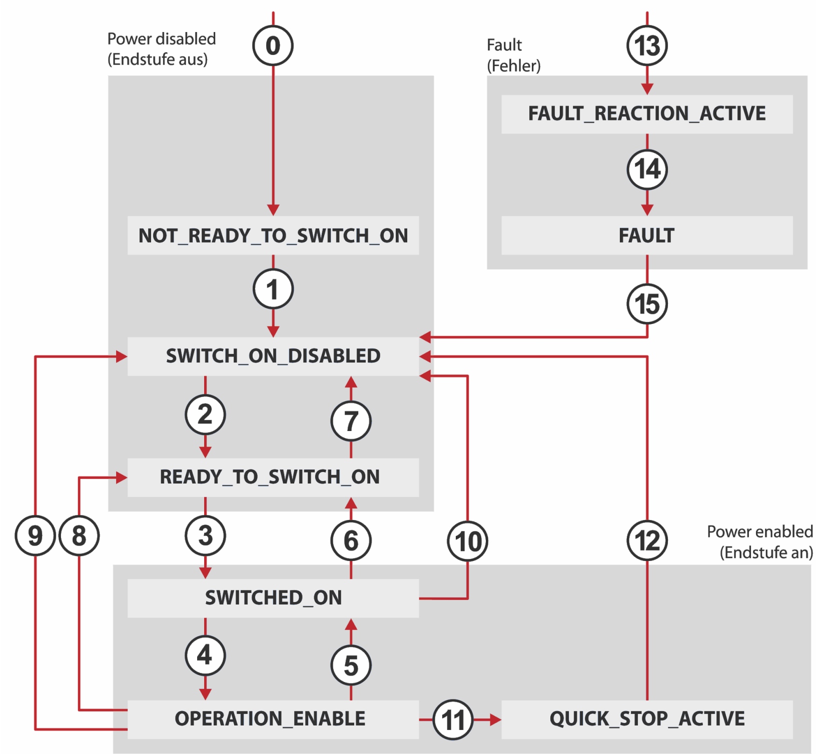


Figure 9: State diagram of the servo drive

After switching on, the servo drive initialises itself and finally reaches the `SWITCH_ON_DISABLED` state. In this state, the CAN communication is fully functional and the servo drive can be parameterised (e.g. the operating mode "speed control" can be set). The output stage is switched off and the motor shaft can therefore rotate freely. By means of the state transitions 2, 3, 4 - which in principle corresponds to the CAN servo drive enable - the `OPERATION_ENABLE` state is reached. In this state, the power stage is switched on and the motor is controlled according to the set operating mode. Therefore, before doing so, make absolutely sure that the drive is correctly parameterised and a corresponding setpoint value is zero. State transition 9 corresponds to disabling the drive, i.e. a motor still running would coast down uncontrolled. If an error occurs, the drive (regardless of its current state) ultimately switches to the `FAULT` state. Depending on the severity of the fault, certain actions, such as emergency braking, can be carried out beforehand (`FAULT_REACTION_ACTIVE`).

To execute the mentioned state transitions certain bit combinations must be set in the controlword (see below). The lower 4 bits of the controlword are evaluated together to trigger a state transition. In the following only the most important state transitions 2, 3, 4, 9 and 15 are explained. A table of all possible states and state transitions can be found at the end of this chapter.

Important state transitions

The following table contains in the 1st column the desired state transition and in the 2nd column the necessary prerequisites for it (usually a command by the host, shown here with a frame). How this command is generated, i.e. which bits are to be set in the controlword, can be seen in the 3rd column (x = not relevant).

No.	Is carried out if	Bit combination (controlword)					Action
		Bit	3	2	1	0	
2	Power stage and Controller enable + Shutdown	Shutdown	x	1	1	0	No action
3	Switch On	Switch On	x	1	1	1	Switching on the power stage
4	Enable Operation	Enable Operation	1	1	1	1	Control according to set operating mode
9	Disable Voltage	Disable Voltage	x	x	0	x	Power stage will be disabled. Motor shaft is freely rotatable.
15	Cause of the error eliminated + Fault Reset	Fault Reset	Bit 7 = ↑				Error acknowledgement

EXAMPLE

After the servo drive has been parameterised, the drive should be enabled, i.e. the power stage should be switched on:

1. The servocontroller is in `SWITCH_ON_DISABLED` state
2. The controller should be set to the `OPERATION_ENABLED` state
3. The state transitions 2, 3 and 4 must be executed.
4. From the previous table follows:

Transition	controlword	New state
2	0006_h	`READY_TO_SWITCH_ON`
3	0007_h	`SWITCHED_ON`
4	$000F_h$	`OPERATION_ENABLE`

Remarks:

- To illustrate the principle, no further bits are set in the controlword.
- The transitions 3 and 4 can be combined by writing $000F_h$, because the set bit 3 is not relevant for transition 3.
- In each case, it is necessary to wait until the controller has reached this state. This is explained in more detail in the following section.

4.2.1 State diagram: States

In the following table all states and their meaning are listed:

Name	Description
NOT_READY_TO_SWITCH_ON	The servo drive performs a self-test. The CAN communication is not yet working.
SWITCH_ON_DISABLED	The servo drive has completed its self-test. CAN communication is possible.
READY_TO_SWITCH_ON	The servo drive waits until the digital inputs "Power stage enable" and "Controller enable" are connected to 24 V. (Enable logic "Digital input and CAN").
SWITCHED_ON *1)	The power stage is switched on.
OPERATION_ENABLE *1)	The motor is supplied with voltage and is controlled according to the current operating mode.
QUICKSTOP_ACTIVE *1)	The Quick Stop Function is executed (see: quick_stop_option_code). The motor is connected to voltage and is controlled according to the Quick Stop Function.
FAULT_REACTION_ACTIVE *1)	An error has occurred. In the case of critical errors, the device immediately switches to the status Fault. Otherwise, the action specified in the fault_reaction_option_code is executed. The motor is connected to voltage and is controlled according to the Fault Reaction Function.
FAULT	An error has occurred. The motor is voltage-free.

*1) The power stage is switched on

4.2.2 State diagram: State transitions

⚠ DANGER ⚠ **Danger to life due to electric shock!**

Power stage disabled means that the power semiconductors are no longer driven. If this state is entered when the motor is rotating, it coasts down unbraked. A mechanical motor brake, if present, is automatically applied.

The signal does not guarantee that the motor is in fact voltage-free.

⚠ CAUTION **Uncontrolled behaviour**

Power stage enabled means that the motor is controlled according to the selected operating mode. A mechanical motor brake, if present, is automatically released.

In the event of a defect or incorrect parameterisation (motor current, number of poles, resolver offset angle, etc.), the drive may behave in an uncontrolled manner.

The following table lists all state transitions and their meaning:

No.	Will be executed if	Bit combination (controlword)					Action
		Bit	3	2	1	0	
0	Switched on or reset	Internal transition					Perform self-test
1	Self-test succesful	Internal transition					Activation of CAN communication
2	Dig. inputs Power stage enable and Controller enable active + Shutdown	Shutdown	x	1	1	0	-
3	Switch On	Switch On	x	1	1	1	Power stage switched on
4	Enable Operation	Enable Operation	1	1	1	1	Control according to set operating mode
5	Disable Operation	Disable Operation	0	1	1	1	Power stage is disabled. Motor shaft is freely rotatable.
6	Shutdown	Shutdown	x	1	1	0	Power stage is disabled. Motor shaft is freely rotatable.
7	Quick Stop	Quick Stop	x	0	1	x	-
8	Shutdown	Shutdown	x	1	1	0	Power stage is disabled. Motor shaft is freely rotatable.
9	Disable Voltage	Disable Voltage	x	x	0	x	Power stage is disabled. Motor shaft is freely rotatable.
10	Disable Voltage	Disable Voltage	x	x	0	x	Power stage is disabled. Motor shaft is freely rotatable.
11	Quick Stop	Quick Stop	x	0	1	x	Braking according to quick_stop_option_code.
12	Braking finished or Disable Voltage	Disable Voltage	x	x	0	x	Power stage is disabled. Motor shaft is freely rotatable.

No.	Will be executed if	Bit combination (controlword)					Action
		Bit	3	2	1	0	
13	Error occurred	Internal transition					For non-critical errors, reaction according to fault_reaction_option_code. For critical errors, transition 14 follows
14	Error handling is finished	Internal transition					Power stage is disabled. Motor shaft is freely rotatable.
15	Cause of error eliminated + Command Fault Reset	Fault Reset	Bit 7 = ↑				Error acknowledgement (on rising edge)

4.3 controlword

Object 6040_h: controlword

The controlword can be used to change the current state of the servo drive or directly trigger a specific action (e.g. start homing). The function of bits 4, 5, 6 and 8 depends on the current operating mode (modes_of_operation) of the servo drive, which is explained after this chapter.

Index	**6040_h**			
Name	**controlword**			
Info	--	rw	PDO	UINT16
Value	--	--		

Bit	Value	Function
0	0001_h	Control of the state transitions. (These bits are evaluated together)
1	0002_h	
2	0004_h	
3	0008_h	
4	0010_h	`new_set_point` / `start_homing_operation` / `enable_ip_mode`
5	0020_h	`change_set_immediatly`
6	0040_h	`absolute / relative`
7	0080_h	`reset_fault`
8	0100_h	`halt`
9	0200_h	Reserved, write 0.
10	0400_h	Reserved, write 0.
11	0800_h	Reserved, write 0.
12	1000_h	Reserved, write 0.
13	2000_h	Reserved, write 0.
14	4000_h	Reserved, write 0.
15	8000_h	Reserved, write 0.

Description of the commands (Bits 0...3, Bit 7)

As already extensively described, state transitions can be executed with bits 0..3. The commands required for this are shown here once again in an overview. The Fault Reset command is generated by a rising edge (from 0 to 1) of bit 7.

Commands:	Bit 7	Bit 3	Bit 2	Bit 1	Bit 0
	0080_h	0008_h	0004_h	0002_h	0001_h
Shutdown	x	x	1	1	0
Switch On	x	x	1	1	1
Disable Voltage	x	x	x	0	x
Quick Stop	x	x	0	1	x
Disable Operation	x	0	1	1	1
Enable Operation	x	1	1	1	1
Fault Reset	rising edge	x	x	x	x

INFORMATION **State changes**

Since some status changes take a certain amount of time, all status changes triggered by the `controlword` must be read back via the `statusword`. Only when the requested status can also be read in the `statusword` is it permitted to write another command into the `controlword`.

Description of the other bits

The remaining bits of the controlword are explained below. Some of the bits have different meanings depending on the operation mode (modes_of_operation), i.e. whether the servo drive is speed- or torque-controlled, for example:

Bit 4	Depends on *modes_of_operation*:
`new_set_point`	In Profile Position Mode: A rising edge signals the servocontroller that a new positioning job should be accepted. See also section 5.3 *Profile Position Mode* on page 134.
`start_homing_operation`	In Homing Mode: A rising edge causes the parameterised homing run to be started. A falling edge terminates an active homing run.
`enable_ip_mode`	In Interpolated Position Mode: This bit must be set to enable the interpolation data sets to be evaluated. It is acknowledged by the bit ip_mode_active in the statusword. For more information see section 5.4 *Interpolated Position Mode* on page 139

Bit 5	
`change_set_immediatly`	Only in Profile Position Mode: If this bit is not set, a possibly running positioning job is processed first and then the new one is started. If this bit is set, a running positioning is immediately aborted and replaced by the new positioning job. It is essential that you also refer to section 5.3 *Profile Position Mode* on page 134.
Bit 6	
`relative`	Only in Profile Position Mode: When this bit is set, the servocontroller adds the target position (target_position) of the current positioning job to the set position (position_demand_value) of the position controller.
Bit 7	
`reset_fault`	On a rising edge the servocontroller attempts to acknowledge the existing errors. This is only successful if the cause of the error has been eliminated.
Bit 8	**Depends on *modes_of_operation*:**
`halt`	In Profile Position Mode: If the bit will be set, the current positioning is aborted. Braking is done with the profile_deceleration. When the process is complete, the bit target_reached is set in the statusword. Clearing the bit has no effect.
`halt`	In Profile Velocity Mode: When the bit will be set, the speed is reduced to zero. Braking is done with the profile_deceleration. Clearing the bit causes the servocontroller to accelerate again.
`halt`	In Profile Torque Mode: When the bit will be set, the torque is reduced to zero. This is done with the torque_slope. Clearing the bit causes the servocontroller to accelerate again.
`halt`	In Homing Mode: When the bit will be set, the current homing run is aborted. Clearing the bit has no effect.

4.4 Reading the servo drive status

In the same way as various state transitions can be triggered by combining several bits of the controlword, the status of the servocontroller can be read out by combining different bits of the statusword. The following table lists the possible states of the state diagram and the corresponding bit combination with which they are displayed in the statusword.

State	Bit 6	Bit 5	Bit 3	Bit 2	Bit 1	Bit 0	Mask	Value
	0040_h	0020_h	0008_h	0004_h	0002_h	0001_h		
Not_Ready_To_Switch_On	0	x	0	0	0	0	**$004F_h$**	**0000_h**
Switch_On_Disabled	1	x	0	0	0	0	**$004F_h$**	**0040_h**
Ready_to_Switch_On	0	1	0	0	0	1	**$006F_h$**	**0021_h**
Switched_On	0	1	0	0	1	1	**$006F_h$**	**0023_h**
Operation_Enable	0	1	0	1	1	1	**$006F_h$**	**0027_h**
Quick_Stop_Active	0	0	0	1	1	1	**$006F_h$**	**0007_h**
Fault_Reaction_Active	0	x	1	1	1	1	**$004F_h$**	**$000F_h$**
Fault	0	x	1	1	1	1	**$004F_h$**	**$000F_h$**
Fault (as per DS402) 1)	0	x	1	0	0	0	**$004F_h$**	**0008_h**

INFORMATION **FAULT state not implemented according to DS402**

1) In earlier CANopen implementations the FAULT state is not indicated according to DS 402. To get the state indicated according to DS 402, this must be selected in the compatibility_control (see section 3.2 *Compatibility settings* on page 41). For compatibility to earlier firmware versions, no changes need to be made.

EXAMPLE

The example on page *104* shows which bits must be set in the controlword to enable the servo drive. In this example, we will explain how the current status of the servo drive is then read from the statusword.

Transition	controlword	New state	Wait until
2	0006_h	READY_TO_SWITCH_ON	(statusword & $006F_h$) = 0021_h
3+4	$000F_h$	OPERATION_ENABLE	(statusword & $006F_h$) = 0027_h

Remarks:

- To clarify the principle, no further bits are set in the controlword.
- To determine the controller status unambiguously, even bits that are not set must be checked in the statusword. The statusword must therefore be masked accordingly.

4.5 Statuswords

4.5.1 Object 6041_h: statusword

Index	**6041_h**			
Name	**statusword**			
Info	--	ro	PDO	UINT16
Value	--	--		

Bit	Value	Name
0	0001_h	Status of the servo drive, see section 4.4 *Reading the servo drive status* on page 111. These bits must be evaluated together.
1	0002_h	
2	0004_h	
3	0008_h	
5	0020_h	
6	0040_h	
4	0010_h	`voltage_enabled`
7	0080_h	`warning`
8	0100_h	`drive_is_moving`
9	0200_h	`remote`
10	0400_h	`target_reached`
11	0800_h	`internal_limit_active`
12	1000_h	`set_point_acknowledge` / `speed_0` / `homing_attained` / `ip_mode_active`
13	2000_h	`following_error` / `homing_error`
14	4000_h	`manufacturer_statusbit`
15	8000_h	`trigger_result`

All bits of the statusword are not buffered. They represent the current device status.

In addition to the servo drive status, various events are displayed in the statusword, whereby each bit is assigned a specific event, such as a following error. The individual bits have the following meaning:

Bit 4	
`voltage_enabled`	This bit is set when the power stage transistors are **switched off**. In earlier CANopen implementations, contrary to the specification in DS 402, bit 4 (`voltage_enabled`) is returned inverted. For compatibility reasons, this has been retained. However, it is possible to select the behaviour according to DS402 via the object compatibility_control (see section 3.2 *Compatibility settings* on page 41). If bit 7 is set in object 6510_h_F0h (compatibility_control), the following applies: This bit is set if the power stage transistors are **switched on**. No changes need to be made for compatibility with earlier firmware versions.
Bit 5	
`quick_stop`	If the bit is cleared, the drive executes a Quick Stop according to quick_stop_option_code.
Bit 7	
`warning`	The meaning of this bit is configurable: It can be set when any bit in manufacturer_warnings_1 is set. See also section 4.5.5 *Object 2001h: manufacturer_warnings* on page 118.
Bit 8	**manufacturer specific**
`drive_is_moving`	This bit is set - independently of the modes_of_operation - if the actual speed (velocity_ actual_value) of the drive is outside the associated tolerance window (velocity_threshold).
Bit 9	
`remote`	This bit indicates that the power stage of the servocontroller can be enabled via the CAN network. It is set if the controller enable logic is set accordingly via the enable_logic object.

Bit 10	Depends on *modes_of_operation:*
`target_reached`	In Profile Position Mode: This bit is set when the target position is reached and the actual position (position_ actual_value) is in the parameterised position window (position_window). It is also set when the drive comes to a standstill after the Halt bit has been set. It is deleted as soon as a new positioning is started.
`target_reached`	In Profile Velocity Mode: The bit is set when the speed (velocity_actual_value) of the drive is within the tolerance window (velocity_window, velocity_ window_time).
Bit 11	
`internal_limit_ active`	This bit indicates that the I^2t limitation is active.
Bit 12	**Depends on *modes_of_operation:***
`set_point_ acknowledge`	In Profile Position Mode: This bit is set when the servocontroller has recognised the set bit new_set_point in the controlword. It is cleared again after the new_set_point bit in the controlword has been set to zero. For more information see section 5.3 *Profile Position Mode* on page 134.
`speed_0`	In Profile Velocity Mode: This bit is set when the actual speed (velocity_actual_ value) of the drive is within the associated tolerance window (velocity_threshold).
`homing_attained`	In Homing Mode: This bit is set if the homing run was completed without errors.
`ip_mode_active`	In Interpolated Position Mode: This bit indicates that the interpolation is active and the interpolation data sets are being evaluated. It is set if this was requested by the bit `enable_ip_mode` in the controlword. For more information see section 5.4 *Interpolated Position Mode* on page 139.

Bit 13	Depends on *modes_of_operation:*
`following_error`	In Profile Position Mode: This bit is set if the actual position (position_actual_ value) differs from the target position (position_ demand_value) so much that the difference lies outside the parameterised tolerance window (following_error_window, following_error_ time_ out).
`homing_error`	In Homing Mode: This bit is set if the homing run is interrupted (Halt bit), both limit switches are activated simultaneously or the distance already travelled during the limit switch search is greater than the specified positioning range (min_position_limit, max_ position_limit).
Bit 14	**manufacturer specific**
`manufacturer_statusbit`	The meaning of this bit is configurable: It can be set when any bit of the manufacturer_statusword_1 is set or reset. See also section 4.5.2 *Object 2000h: manufacturer_statuswords* on page 115.
Bit 15	**manufacturer specific**
`trigger_result`	The meaning of this bit is configurable: It is set when a sample event has occurred and the sample mask is set accordingly. See also section 3.15 *Position capturing (Sampling)* on page 89.

4.5.2 Object 2000$_h$: manufacturer_statuswords

The object group manufacturer_statuswords displays additional manufacturer-specific states of the servocontroller.

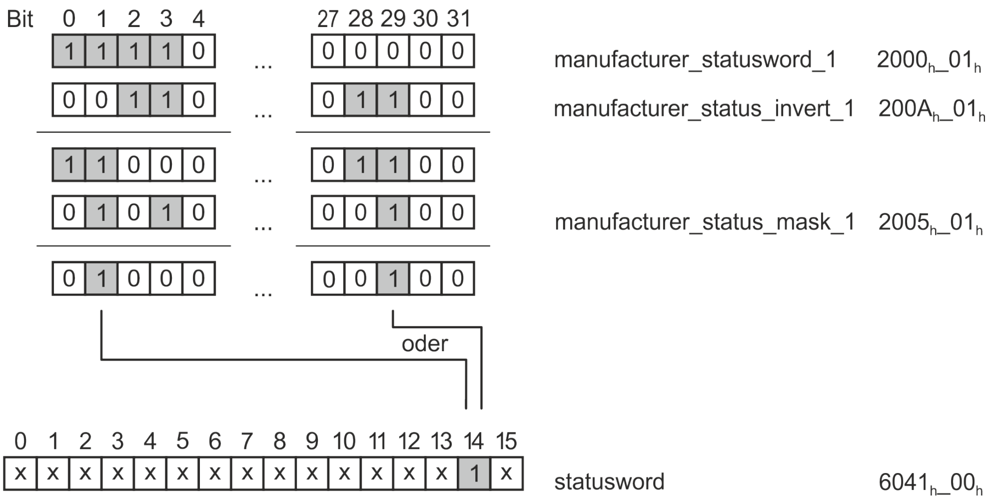
Index	**2000$_h$**	
Name	**manufacturer_statuswords**	
Type	RECORD	01$_h$

Sub-Index	**01$_h$**			
Name	**manufacturer_statusword_1**			
Info	--	ro	PDO	UINT32
Value	--	--		

Bit	Value	Name
0	00000001$_h$	`is_referenced`
1	00000002$_h$	`commutation_valid`
2	00000004$_h$	`ready_for_enable`
3	00000008$_h$	`ipo_in_target`
...		
8	00000100$_h$	`safe_standstill`

Bit 0	
`is_referenced`	The bit is set when the servocontroller is referenced. This is the case if either a homing run has been successfully performed or no homing run is necessary due to the connected encoder system (e.g. in the case of an absolute encoder).
Bit 1	
`commutation_valid`	The bit is set if the commutation information is valid. It is especially helpful for encoder systems without commutation information (e.g. linear motors), as the automatic commutation finding can take some time there. If this bit is monitored a timeout of the control can be prevented when enabling the servo drive.
Bit 2	
`ready_for_enab`	The bit is set when all conditions are present to enable the servo drive and only the controller enable itself is missing. The following conditions must be fulfilled: • The drive is error-free • The DC link is loaded • The angle encoder evaluation is ready. No processes (e.g. serial transmission) are active that prevent an enable. • No blocking process is active (e.g. automatic motor parameter identification)
Bit 3	
`ipo_in_target`	The bit is set when the trajectory generator has completed the positioning. In contrast to `target_reached`, no additional check is made whether the actual position also corresponds to the target position.
Bit 8	
`safe_standstill`	The bit is set when the controller has entered the safe state "Safe Torque Off" (STO). See also the relevant section in the product manual, e.g. section STO (*Safe Torque Off*) in the BL 4000-C product manual.

With the help of the objects manufacturer_status_masks and manufacturer_status_invert one or more bits of the manufacturer_statuswords can be mapped into bit 14 (`manufacturer_statusbit`) of the statusword (6041_h). All bits of the manufacturer_statusword_1 can be inverted via the corresponding bit in manufacturer_status_invert_1. Thus, bits can also be monitored for the "reset" status. After the inversion the bits are masked, i.e. only if the corresponding bit in manufacturer_status_mask_1 is set, the bit is further evaluated. If at least one bit is still set after masking, bit 14 of the statusword is also set. The following figure illustrates this as an example:



EXAMPLE

Bit 14 of the statusword should be set if the drive is referenced:

Object	Value	
manufacturer_status_invert_1	0x00000000	Invert no bit
manufacturer_status_mask_1	0x00000001	Show bit 0

Bit 14 of the statusword should be set if the drive has **no** valid commutation position:

Object	Value	
manufacturer_status_invert_1	0x00000002	Invert bit 1
manufacturer_status_mask_1	0x00000002	Show bit 1

Bit 14 of the statusword should be set if the drive is not ready for enable OR referenced:

Object	Value	
manufacturer_status_invert_1	0x00000004	Invert bit 2
manufacturer_status_mask_1	0x00000005	Show bit 0 and bit 2

4.5.3 Object 2005$_h$: manufacturer_status_masks

This object group is used to specify which set bits of the manufacturer_statuswords are mapped into the statusword.

Index	**2005$_h$**			
Name	**manufacturer_status_masks**			
Type	RECORD			01$_h$

Sub-Index	**01$_h$**			
Name	**manufacturer_status_mask_1**			
Info	--	rw	PDO	UINT32
Value	--	0		

4.5.4 Object 200A$_h$: manufacturer_status_invert

This object group determines which bits of the manufacturer_statuswords are inverted before masking.

Index	**200A$_h$**			
Name	**manufacturer_status_invert**			
Type	RECORD			01$_h$

Sub-Index	**01$_h$**			
Name	**manufacturer_status_invert_1**			
Info	--	rw	PDO	UINT32
Value	--	0		

4.5.5 Object 2001$_h$: manufacturer_warnings

The manufacturer-specific object group manufacturer_warnings shows further states of the servo drive.

Index	**2001$_h$**			
Name	**manufacturer_warnings**			
Type	RECORD			01$_h$

Sub-Index	**01$_h$**			
Name	**manufacturer_warnings_1**			
Info	--	ro	PDO	UINT32
Value	--	--		

Bit	Value	Name
0	00000001$_h$	`l_lim_switch_lock`
1	00000002$_h$	`r_lim_switch_lock`
2	00000004$_h$	`warning_active`

Bit 0	
`l_lim_switch_lock`	This bit indicates that the direction is locked because the left limit switch has been triggered. The setpoint lock is reset when an error acknowledgement is performed (See controlword, `fault_reset`).
Bit 1	
`r_lim_switch_lock`	This bit indicates that the direction is locked because the right limit switch has been triggered. The setpoint lock is reset when an error acknowledgement is performed (See controlword, `fault_reset`).
Bit 2	
`warning_active`	This bit indicates that a warning is active in the servo drive, see the corresponding section in the product manual, e.g. section *Fault messages* in the BL 4000 Product manual.

With the help of the manufacturer_warning_masks object, one or more bits of the manufacturer_warnings can be mapped into bit 7 (`warning`) of the statusword (6041_h). Only if the corresponding bit in manufacturer_warning_mask_1 is set, the bit is further evaluated. If at least one bit is still set after masking, bit 7 of the statusword is also set.

4.5.6 Object 2006_h: manufacturer_warning_masks

This object group determines which set bits of the manufacturer_warnings object are mapped into the statusword.

Index	**2006_h**	
Name	**manufacturer_warning_masks**	
Type	RECORD	01_h

Sub-Index	**01_h**			
Name	**manufacturer_warning_mask_1**			
Info	--	rw	PDO	UINT32
Value	--	0		

4.6 Description of further objects

4.6.1 Object 605Bh: shutdown_option_code

The shutdown_option_code is used to specify how the servocontroller behaves during state transition 8 (from OPERATION_ENABLE to READY_TO_SWITCH_ON). The object indicates the unchangeable behavior of the servocontroller.

Index	**605Bh**			
Name	**shutdown_option_code**			
Info	--	rw	~~PDO~~	INT16
Value	0	--		

Value	Name
0	Output stage will be switched off, motor can rotate freely

4.6.2 Object 605Ch: disable_operation_option_code

The disable_operation_option_code object is used to specify how the servocontroller behaves during state transition 5 (from OPERATION_ENABLE to SWITCHED_ON). The object indicates the unchangeable behavior of the servocontroller.

Index	**605Ch**			
Name	**disable_operation_option_code**			
Info	--	rw	~~PDO~~	INT16
Value	-1	--		

Value	Name
-1	Decelerate with quickstop_deceleration

4.6.3 Object 605Ah: quick_stop_option_code

The Parameter quick_stop_option_code is used to specify how the servocontroller behaves in the event of a Quick Stop. The object indicates the unchangeable behavior of the servocontroller.

Index	**605Ah**			
Name	**quick_stop_option_code**			
Info	--	rw	~~PDO~~	INT16
Value	2	--		

Value	Name
2	Decelerate with quickstop_deceleration

4.6.4 Object 605E$_h$: fault_reaction_option_code

The fault_reaction_option_code object is used to specify how the servo drive behaves in the event of a fault. Since the error reaction depends on the respective error, this object cannot be parameterised and always returns 0. To change the error reaction of the individual errors see section 3.18 *Error management* on page 100.

Index	**605E$_h$**			
Name	**fault_reaction_option_code**			
Info	--	rw	~~PDO~~	INT16
Value	0	--		

5 Operating modes

5.1 Setting the operating mode

5.1.1 Overview

The servocontroller can be set to a variety of operating modes. Only a few are specified in detail under CANopen:

- torque-controlled operation (profile torque mode)
- speed-controlled operation (profile velocity mode)
- homing mode
- positioning mode (profile position mode)
- synchronous position mode (CANopen: interpolated position mode, Ethercat: cyclic synchronous position mode)

5.1.2 Description of objects

5.1.2.1 Object 6060_h: modes_of_operation

The modes_of_operation object is used to set the operating mode of the servocontroller.

Index	**6060_h**			
Name	**modes_of_operation**			
Info	--	rw	PDO	INT8
Value	1, 3, 4, 6, 7, 8	--		

Value	Action
1	Profile Position Mode (Position control with positioning mode)
3	Profile Velocity Mode (Speed control with setpoint ramp)
4	Profile Torque Mode (Torque control with setpoint ramp)
6	Homing Mode (Reference run)
7	Interpolated Position Mode
8	Cyclic Synchronous Position Mode

INFORMATION **Current operating mode**

The current operating mode can only be read from the object modes_of_operation_display. Since changing the operating mode can take some time, you must wait until the newly selected mode appears in the object modes_of_operation_display.

5.1.2.2 Object 6061h: modes_of_operation_display

The current operating mode of the servocontroller can be read with object modes_of_operation_display.

Index	**6061_h**			
Name	**modes_of_operation_display**			
Info	--	ro	PDO	INT8
Value	-14, -13, -11, -1, 1, 3, 4, 6, 7, 8	--		

If an operating mode is set via object 6060_h, in addition to setting the actual operating mode, the setpoint selector is also modified as follows to ensure operation of the servocontroller under CANopen:

Selector	Profile Velocity Mode	Profile Torque Mode
A	Speed setpoint (Fieldbus 1)	Torque setpoint (Fieldbus 1)
B	Torque limitation, if applicable	inactive
C	Speed setpoint (synchronous speed)	inactive

In addition, the setpoint ramp will always be switched on. Only if these settings are made in the mentioned way, one of the CANopen operating modes is displayed. If these settings are changed e.g. with the MPC ServoCommander , the appropriate "User" mode is displayed to indicate that the selectors have been changed.

Value	Mode
-1	Unknown operating mode / operating mode change
-11	User Position Mode
-13	User Velocity Mode
-14	User Torque Mode
1	Profile Position Mode
3	Profile Velocity Mode
4	Torque Profile Mode
6	Homing Mode
7	Interpolated Position Mode
8	Cyclic Synchronous Position Mode

INFORMATION **Setting the operating mode**

The operating mode can only be set via object modes_of_operation. Since changing the operating mode can take some time, you have to wait until the newly selected mode appears in object modes_of_operation_display. During this period, "invalid operating mode" (-1) may be displayed briefly.

5.2 Homing Mode

5.2.1 Overview

This chapter describes how the servo drive searches for the initial position (also called reference point or zero point). There are different methods to determine this position, either the limit switches at the end of the positioning range can be used or a reference switch (zero point switch) within the possible positioning range. In order to achieve the greatest possible reproducibility, the zero pulse of the angle encoder used (resolver, incremental encoder, etc.) can be included in some methods.

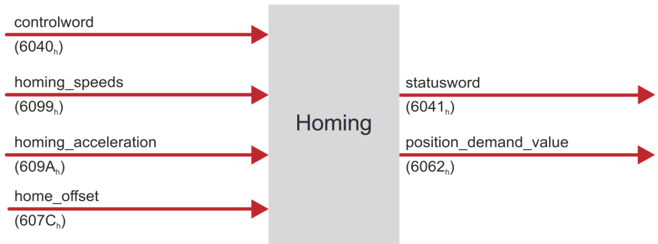


Figure 10: Homing

The user can determine the speed, acceleration and type of homing. The home_offset object can be used to move the zero position of the servo drive to any position. There are two homing speeds. The higher search speed (speed_during_search_for_switch) is used to find the limit switch or the reference switch. In order to subsequently be able to determine the position of the relevant switching edge exactly, the crawling speed (speed_during_search_for_zero) will be used. The maximum distance searched for switches is calculated from the difference of objects $607D_h$_01_h und $607D_h$_02_h (see section 3.7.2.14 *Object 607Dh: software_position_limit* on page 70). If no switch is found within this distance, error 11-6 (Homing: end of search distance reached) is triggered.

INFORMATION **Homing behaviour can be parameterised**

The following homing behaviour can be modified:

- If the reference run is started via the controlword, the servo drive does not necessarily move to the zero position after the reference run. If the servo drive knows all the required values (e.g. because it already knows the position of the zero pulse), no physical movement is carried out.
- The maximum search distance is determined by object $607D_h$.

If bit 6 of the object 6510_h_$F0_h$ (compatibility_control) is set, the settings defined in the MPC ServoCommander are used instead (siehe section 3.2 *Compatibility settings* on page 41).

If the drive should not be referenced, but only the position should be set to a certain value, object 2030_h (set_position_absolute) can be used. For this see section 3.7.2.13 *Object 2030h: set_position_absolute* on page 69.

5.2.2 Description of objects

5.2.2.1 Important objects in other sections

Index	Name	Section	Page
6040_h	controlword	Device Control	*102*
6041_h	statusword		

5.2.2.2 Object $607C_h$: home_offset

The home_offset object specifies the offset of the zero position with respect to the determined reference position. The effect of this object can be customised. See also section section 3.2.2.1 *Object 6510h_F0h: compatibility_control* on page 41.

Index	**$607C_h$**				
Name	**home_offset**				
Info	position_unit	rw	PDO	INT32	
Value	--	--			

5.2.2.3 Object 6098_h: homing_method

A number of different methods are provided for a homing run. The variant required for the application can be selected via the homing_method object. There are four possible homing signals: the negative and positive limit switches, the reference switch and the (periodic) zero pulse of the angle encoder.

In addition, the servo drive can reference to the negative or positive stop without any additional signal at all. If a method for referencing is set via the object homing_method, the following settings are determined with this:

- The reference source (neg./pos. limit switch, the reference switch, neg. / pos. stop).
- The direction and the sequence of the homing
- The method of evaluation of the zero pulse from the used angle encoder

Index	**6098_h**			
Name	**homing_method**			
Info	--	rw	PDO	INT8
Value	-18, -17, -2, -1, 1, 2, 7, 11, 17, 18, 23, 27, 32, 33, 34, 35	--		

	Direction	Target	Reference point for zero	DS402
-18	positive	Stop	Stop	-18
-17	negative	Stop	Stop	-17
-2	positive	Stop	Zero pulse	-2
-1	negative	Stop	Zero pulse	-1
1	negative	Limit switch	Zero pulse	1
2	positive	Limit switch	Zero pulse	2
7	positive	Reference switch	Zero pulse	7
11	negative	Reference switch	Zero pulse	11
17	negative	Limit switch	Limit switch	17
18	positive	Limit switch	Limit switch	18
23	positive	Reference switch	Reference switch	23
27	negative	Reference switch	Reference switch	27
32	negative	Zero pulse	Zero pulse	**33**
33	positive	Zero pulse	Zero pulse	**34**
34		No movement	Current actual position	**35**

INFORMATION **Homing methods not assigned according to DS402**

In previous CANopen implementations the homing methods 32, 33, 34 and 35 are not assigned according to DS402. Therefore, it is possible to select the assignment according to DS402 via object compatibility_control (siehe section 3.2 *Compatibility settings* on page 41). In this case, the method numbers in the column "DS402" must be used.

For compatibility with previous versions, no changes need to be made and the previous numbers can be used.

The homing_method can only be changed if homing is not active. Otherwise the error message 08 00 00 22h is returned.The sequence of the individual methods is explained in detail in section 5.2.3 *Homing sequences* on page 128.

5.2.2.4 Object 6099_h: homing_speeds

This object determines the speeds used during homing.

Index	**6099_h**	
Name	**homing_speeds**	
Type	ARRAY	02_h

Sub-Index	**01_h**			
Name	**speed_during_search_for_switch**			
Info	speed_unit	rw	PDO	UINT32
Value	--	--		

Sub-Index	**02_h**			
Name	**speed_during_search_for_zero**			
Info	speed_unit	rw	PDO	UINT32
Value	--	--		

INFORMATION **Setting bit 6 in the object compatibility_control**

If bit 6 in object compatibility_control, (siehe section 3.2 *Compatibility settings* on page 41) is set, a movement to zero can be carried out after the homing, for example.

If this bit is set and object speed_during_search_for_switch is written, the speed for searching the switch as well as the speed for moving to zero are written together.

5.2.2.5 Object $609A_h$: homing_acceleration

This object specifies the acceleration used for all acceleration and deceleration processes during the homing run.

Index	**$609A_h$**			
Name	**homing_acceleration**			
Info	acceleration_unit	rw	PDO	UINT32
Value	--	--		

5.2.2.6 Object 2045_h: homing_timeout

The homing run can be monitored for its maximum execution time. For this purpose, the maximum execution time can be specified with the homing_timeout object. If this time is exceeded without the homing run being completed, error 11-3 is triggered. If 0 is written to the object, monitoring is deactivated.

Index	**2045_h**			
Name	**homing_timeout**			
Info	ms	rw	~~PDO~~	UINT16
Value	0, 1 … 65535	--		

5.2.3 Homing sequences

5.2.3.1 Methods -17 and -18: Stop

If this method is used, the drive moves in the positive direction (-18) or negative direction (-17) until it reaches the stop. Normally, a 50% increase of the i²t value is used as the criterion for detecting the stop. Alternatively, a comparison torque value at which the stop will be considered as detected can be specified (see section *Tab: Torques* in the respective product manual). The mechanical design of the stop must be such that it cannot be damaged with the parameterised maximum current. The home position refers directly to the stop. Since, in this case, the home position would be located directly at the stop, the parameter Offset start position should be used to shift the home position in a suitable manner.

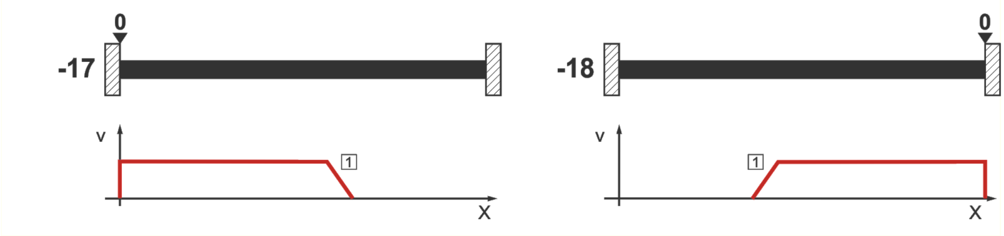


Figure 11: Homing run to the stop

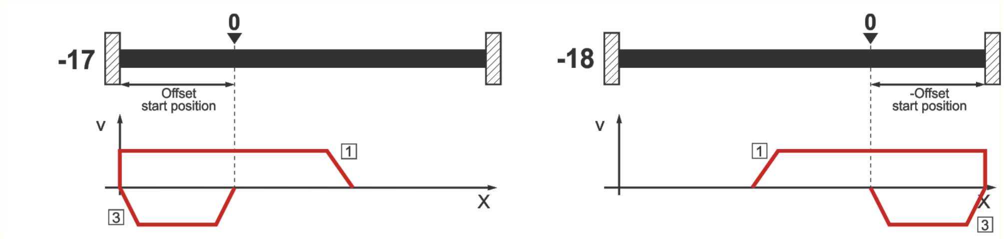


Figure 12: Use of "Offset start position"

5.2.3.2 Methods -1 and -2: stop with index pulse evaluation

These methods correspond to the methods -17 and -18. However, the home position also refers to the first index pulse of the angle encoder in the negative (-2) or positive (-1) direction as seen from the stop.

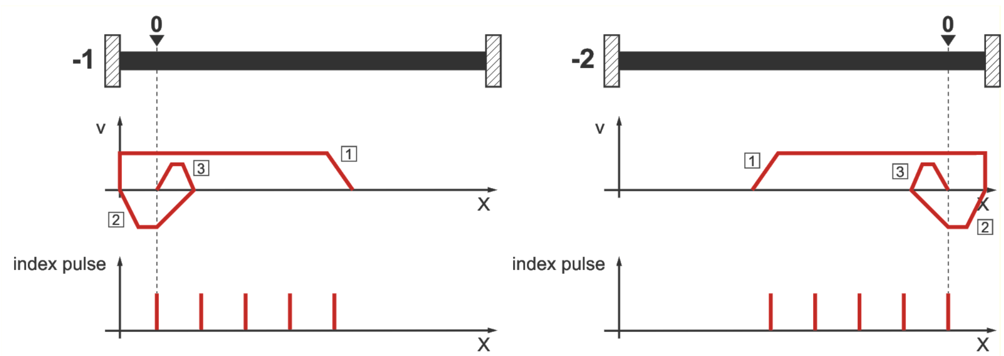


Figure 13: Homing run to the stop with index pulse evaluation

5.2.3.3 Methods 17 and 18: positive and negative limit switch

If these methods are used, the drive moves in the positive direction (18) or negative direction (17) at search speed until it reaches the limit switch. Then, the drive moves back at crawl speed and tries to find the exact position of the limit switch. The home position refers to the falling edge of the limit switch.

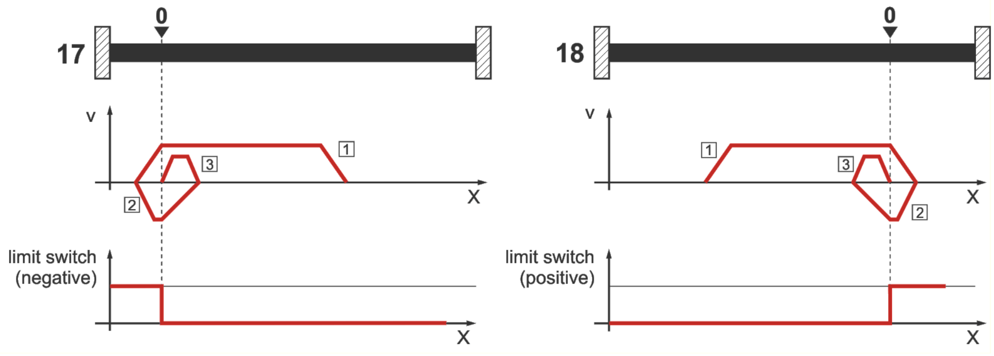


Figure 14: Homing run to the limit switch

5.2.3.4 Methods 1 and 2: positive and negative limit switch with index pulse evaluation

Like in the case of the previous method, the system tries to find the limit switch. However, in this case, the home position refers to the first index pulse of the angle encoder in the negative (1) or positive (2) direction as seen from the limit switch.

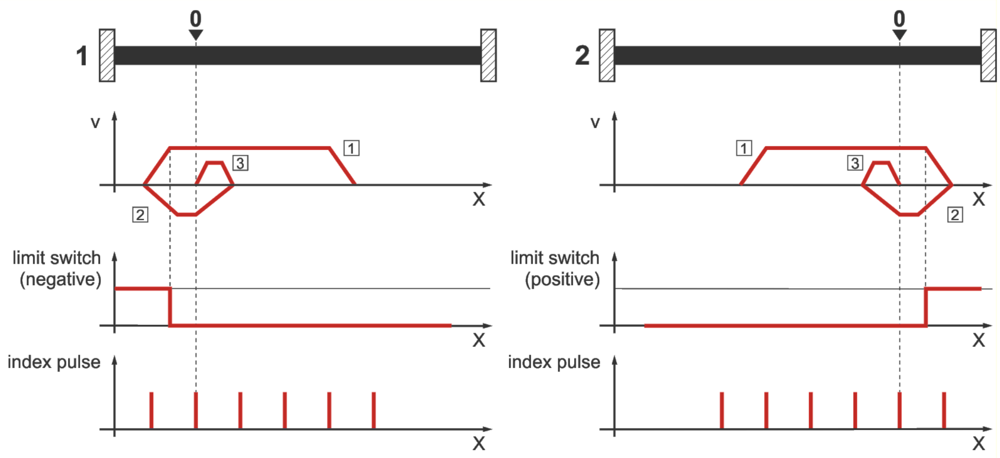


Figure 15: Homing run to the limit switch with index pulse evaluation

5.2.3.5 Methods 23 and 27: reference switch

These two methods use a reference switch which is active only over a certain part of the distance. This method is particularly suitable for rotary axis applications in which the reference switch is activated once during every rotation. If this method is used, the drive moves in the positive direction (23) or negative direction (27) at search speed until it reaches the reference switch. Then, the drive moves back at crawl speed and tries to find the exact position of the reference switch. The home position refers to the falling edge of the reference switch. If, at the beginning, the drive moves away from the reference switch, the associated limit switch causes a reversal of the direction of rotation so that the reference switch will be found.

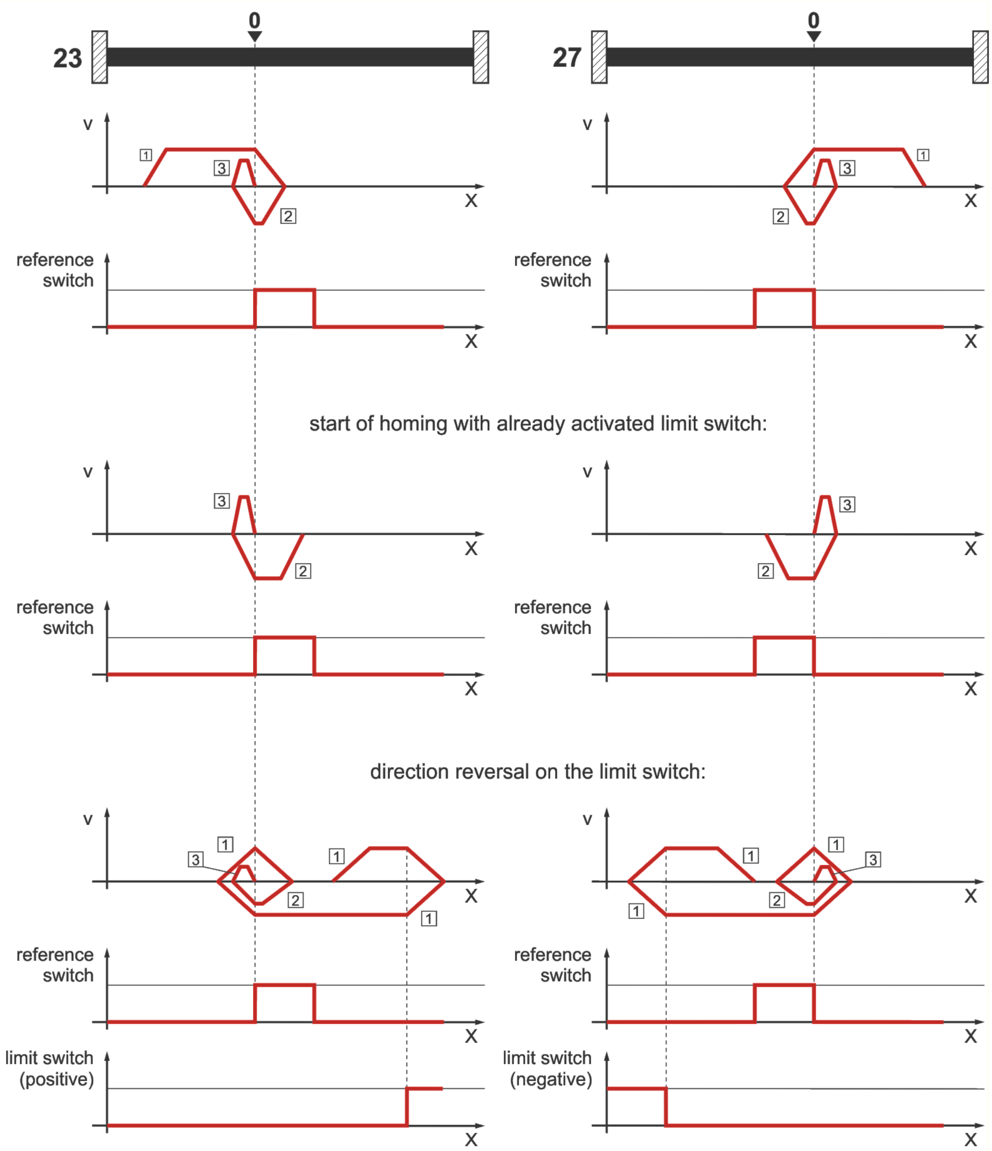


Figure 16: Homing run to the reference switch

5.2.3.6 Methods 7 and 11: reference switch and index pulse evaluation

Like methods 23 and 27, methods 7 and 11 use the reference switch. In addition, however, the home position refers to the first index pulse in the negative or positive direction as seen from the reference switch.

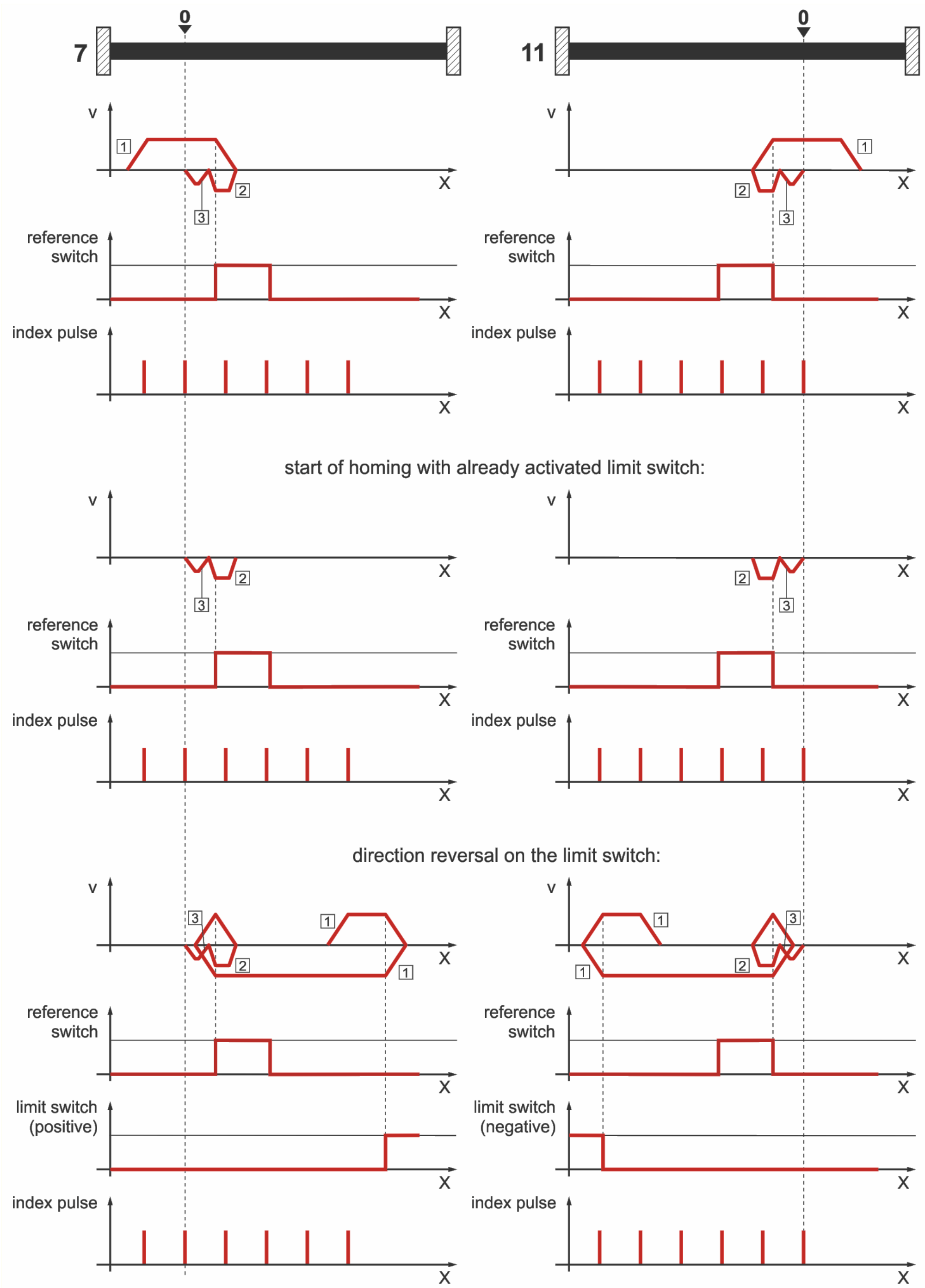


Figure 17: Homing run to the reference switch with index pulse evaluation

5.2.3.7 Methods -23 and -27: homing run (positive/negative) to the reference switch

These methods are similar to the methods 23 and 27. However, in this case, the system tries to locate the end of the range of movement, e.g. the stop or a limit switch, in a first step. It is only then that the system searches for the reference switch. As a result, several switches can be connected to the same input for the reference switch. During the homing run, the "last" switch in the search direction will be used as the reference switch. In the case of method -23, the drive moves in the positive direction first, and in the case of method -27, it moves in the negative direction first. The home position refers to the falling edge of the reference switch.

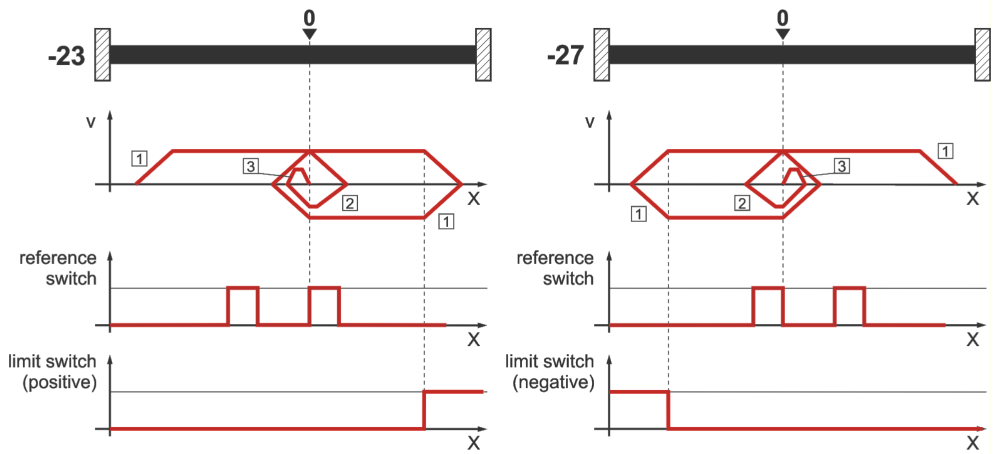


Figure 18: Reference switch with an initial movement in the positive and negative direction

5.2.3.8 Methods 32 and 33: homing to the index pulse

In the case of method 32 and method 33, the direction of the homing run is negative or positive. The home position refers to the first index pulse of the angle encoder in the search direction.

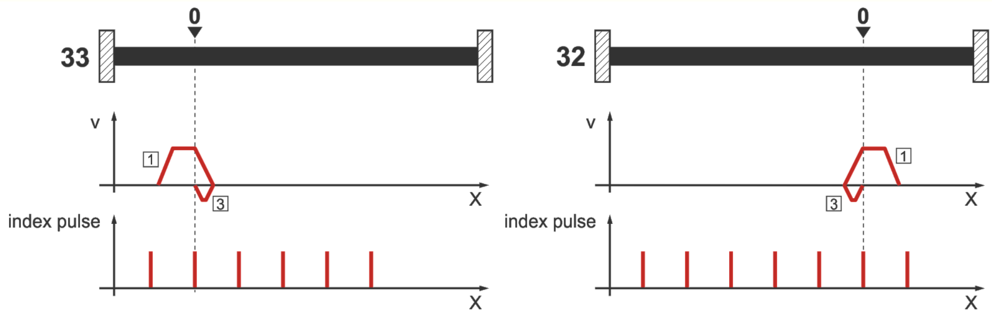


Figure 19: Index pulse with a negative (32) and positive (33) initial movement

5.2.3.9 Method 34: homing to the current position

In the case of method 34, the home position refers to the current position, i.e. the current position of the drive is set to zero.

5.2.4 Homing control

The reference run is controlled and monitored by the controlword / statusword. Starting is done by setting bit 4 in the controlword.

Bit 4	Description
0	Homing is not active
0 ► 1	Start homing
1	Homing is active
1 ◄ 0	Interrupt homing

Successful completion of the homing is indicated by a set bit 12 in the statusword A set bit 13 in the statusword indicates that an error occurred during the reference run. The cause of the error can be determined via the error_register and pre_defined_error_field objects.

Bit 13	Bit 12	Description
0	0	Reference run is not yet ready
0	1	Reference run carried out successfully
1	0	Reference run not carried out successfully
1	1	Illegal state

5.3 Profile Position Mode

5.3.1 Overview

The structure of this operating mode can be seen in *Figure 20: Trajectory generator and position controller*:

The target_position is transferred to the trajectory generator. This generates a position setpoint (position_demand_value) for the position controller, which is described in the Position Controller section (see section 3.7 *Position Controller* on page 63). These two function blocks can be set independently of each other.

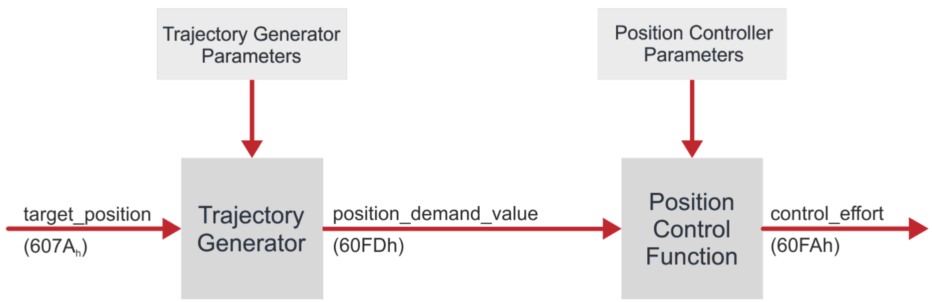


Figure 20: Trajectory generator and position controller

5.3.2 Functional description

There are two ways to transfer a target position to the servo drive:

› Single driving task

When the servo drive has reached a target position, it signals this to the host with the `target_reached` bit (bit 10 in the statusword object). In this operating mode, the servo drive stops when it has reached the target.

› Sequence of driving tasks

After the servo drive has reached a target, it immediately starts moving to the next target. This transition can be carried out smoothly without the servo drive coming to a standstill in between.

These two methods are controlled by the `new_set_point` and `change_set_immediatly` bits in the controlword object and `set_point_acknowledge` in the statusword object. These bits are in a question-answer relationship to each other. This makes it possible to prepare one motion task while another is still running.

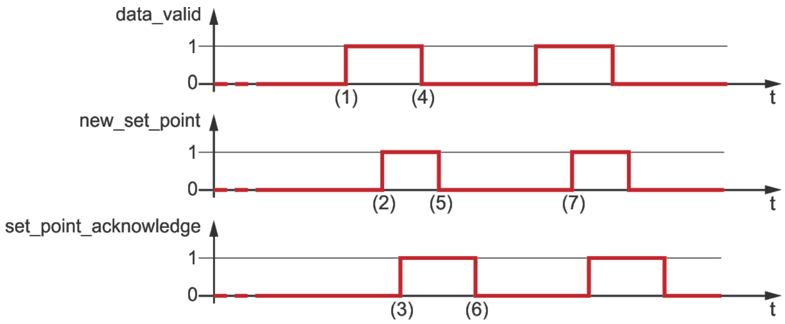


Figure 21: Driving task transfer from a host

The figure above shows how the host and the servo drive communicate with each other via the CAN bus:

First, the positioning data (target position, profile velocity, end velocity and the acceleration) are transmitted to the servo drive. When the positioning data set is completely written (1), the host can start the positioning by setting the bit `new_set_point` in the controlword to "1" (2). After the servo drive has recognised the new data and accepted it into its buffer, it reports this to the host by setting the bit `set_point_acknowledge` in the statusword (3).

The host can then start writing a new positioning data set into the servo drive (4) and clear the `new_set_point` bit again (5). Only when the servo drive can accept a new motion task (6), it signals this by a "0" in the `set_point_acknowledge` bit. Before that, no new positioning may be started by the host (7).

On the left side of the following figure, a new positioning is started only after the previous one has been completed. The host evaluates the target_reached bit in the statusword object for this purpose.

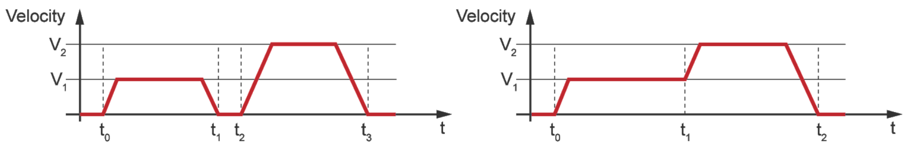


Figure 22: Single driving task (left) and sequence of driving tasks (right)

On the right side, a new positioning is already started while the previous one is still being processed. The host transfers the subsequent target to the servo drive as soon as it signals that it has read the buffer and started the associated positioning by clearing the set_point_acknowledge bit. In this way, the positioning operations are linked together seamlessly. To prevent the servo drive from briefly decelerating to zero each time between the individual positionings, the object end_velocityshould be written with the same value as the object profile_velocity for this operating mode.

If the bit `change_set_immediately` is set to "1" in the controlword in addition to the bit `new_set_point`, the host thereby instructs the servo drive to start the new motion task immediately. A motion task that is already being processed is canceled in this case.

5.3.3 Description of objects

5.3.3.1 Important objects in other sections

Index	Name	Section	Page
6040_h	controlword	*4 Device Control*	*102*
6041_h	statusword		
$605A_h$	quick_stop_option_ code		
$607E_h$	polarity	*3.3 Factor Group*	*43*
6093_h	position_factor		
6094_h	velocity_encoder_ factor		
6097_h	acceleration_factor		

5.3.3.2 Object $607A_h$: target_position

The object target_position determines to which position the servo drive should move. The current setting of the speed, acceleration, deceleration and the type of motion profile (motion_profile_type) must be taken into account. The target position (target_ position) is interpreted either as an absolute or relative value (controlword, Bit 6).

Index	**$607A_h$**			
Name	**target_position**			
Info	position_unit	rw	PDO	INT32
Value	--	--		

5.3.3.3 Object 6081_h: profile_velocity

The profile_velocity object specifies the velocity that is normally reached during a positioning at the end of the acceleration ramp. The profile_velocity object is specified in speed_unit.

Index	**6081_h**			
Name	**profile_velocity**			
Info	speed_unit	rw	PDO	UINT32
Value	--	--		

5.3.3.4 Object 6082_h: end_velocity

The end_velocity object defines the velocity that the drive must have when it reaches the target_position. Normally this object is to be set to zero so that the servo drive stops when it reaches the target position. For gapless positioning, a velocity other than zero can be specified. The end_velocity object is specified in the same unit as the profile_velocity object.

Index	**6082_h**			
Name	**end_velocity**			
Info	speed_unit	rw	PDO	UINT32
Value	--	--		

5.3.3.5 Object 6083_h: profile_acceleration

The profile_acceleration object specifies the acceleration used to accelerate to the setpoint. It is specified in user-defined unit (acceleration_unit).

Index	**6083_h**			
Name	**profile_acceleration**			
Info	acceleration_unit	rw	PDO	UINT32
Value	--	--		

5.3.3.6 Object 6084_h: profile_deceleration

The profile_deceleration object specifies the acceleration with which braking is performed. It is specified in user-defined unit (acceleration_unit).

Index	**6084_h**			
Name	**profile_deceleration**			
Info	acceleration_unit	rw	PDO	UINT32
Value	--	--		

5.3.3.7 Object 6085_h: quick_stop_deceleration

The quick_stop_deceleration object specifies the deceleration with which the motor stops when a quick stop is executed (see section 4.2.2 *State diagram: State transitions* on page 105). The quick_stop_deceleration object is specified in the same unit as the profile_deceleration object.

Index	**6085_h**			
Name	**quick_stop_deceleration**			
Info	acceleration_unit	rw	PDO	UINT32
Value	--	--		

5.3.3.8 Object 6086_h: motion_profile_type

The motion_profile_type object is used to select the type of positioning profile.

Index	**6086_h**			
Name	**motion_profile_type**			
Info	--	rw	PDO	INT16
Value	0, 2	--		

Value	Profile
0	Linear profile
2	Jerkfree profile

5.4 Interpolated Position Mode

5.4.1 Overview

In Interpolated Position Mode (IP), the servo drive follows cyclical position setpoints, e.g. in a multi-axis application of the servo drive. For this purpose, synchronization telegrams (SYNC) and position setpoints are given by a superordinate control in a fixed time grid (synchronization interval, t_P). Since the interval is usually greater than one position control cycle (t_X), the servo drive interpolates the data values between two specified position values, as outlined in the following graphic.

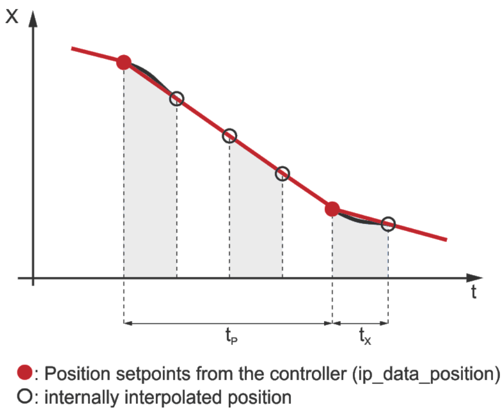


Figure 23: Linear interpolation between two data values

In the following, the objects required for the Interpolated Position Mode are described first. In a subsequent functional description, the activation and the sequence of the parameterization are dealt with comprehensively.

5.4.2 Functional description

Before the servo drive can be switched to Interpolated Position Mode, various settings must be made: These include the setting of the interpolation interval (interpolation_time_period), i.e. the time between two SYNC telegrams, the interpolation type (interpolation_submode_select) and the type of synchronization (interpolation_sync_definition). In addition, access to the position buffer must be enabled via the object buffer_clear. To change the interpolation interval (cycle time), the parameter set must be saved once and the servo drive restarted. Whether the correct interval is set can be read out via the object synchronous_window_length (1006_h). If the correct interval is already set, the first four steps in the following example can be omitted.

EXAMPLE

The example shows which steps are necessary to prepare the servo drive for interpolation operation:

Task	Action
Set time unit (1/10 ms)	$60C2_h$_02_h (interpolation_time_index) = -4
Set time interval (2 ms)	$60C2_h$_01_h (interpolation_time_units) = 20
Save parameters	1010_h_01_h (save_all_parameters) = 65766173_h
Execute reset	see section *6.6 Network Management (NMT service)*
Wait for reboot	see section *6.7 Bootup*
Set type of interpolation	$60C0_h$ (interpolation_submode_select) = -2
Release buffer	$60C4_h$_06_h (buffer_clear) = 1
Start sending SYNC messages	see section *6.5 SYNC message*

The further steps are described in the following sections.

The Interpolated Position Mode is activated via the object modes_of_operation (6060_h). From this point on, the servo drive attempts to synchronise itself to the external time grid, which is specified by the SYNC telegrams. If the servo drive was able to synchronise successfully, it reports the Interpolated Position Mode in the object modes_of_operation_display (6061_h). During synchronization, the servo drive returns "Invalid operation mode". If the SYNC telegrams are not sent in the correct interval after the synchronization has been completed, the servo drive reports "Invalid operating mode" again.

If the change of the operating mode is completed, the transmission of position data to the drive can start. For this purpose, the superordinate control first reads the current actual position from the servo drive and writes it cyclically as the new setpoint (interpolation_data_record) to the servo drive. The handshake bits of the controlword and the statusword are used to activate the acceptance of the data by the servo drive. By setting the bit `enable_ip_mode` in the controlword the host indicates that the evaluation of the position data is to be started. The data sets are not evaluated until the servo drive acknowledges this via the `ip_mode_active` status bit in the statusword. In detail therefore the following sequence results:

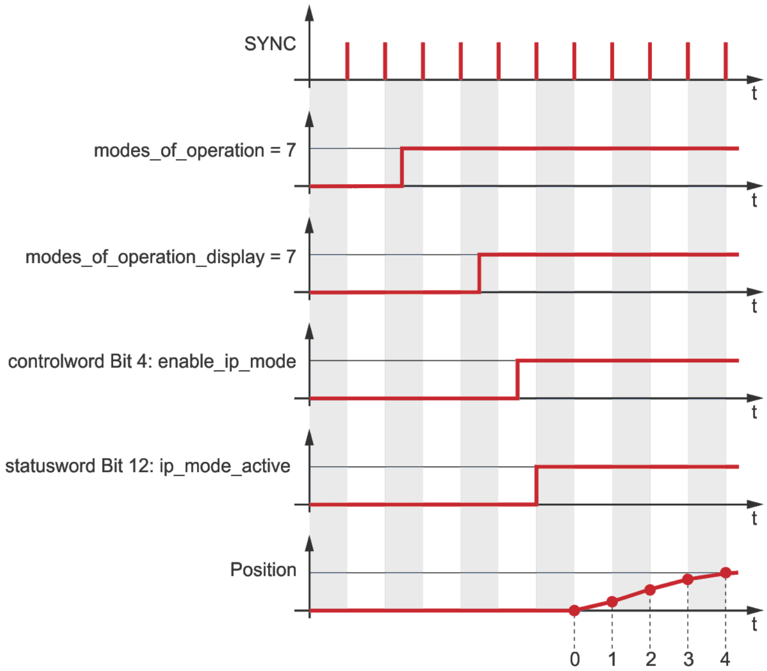


Figure 24: Start of synchronization and data enabling

EXAMPLE

The example shows which steps are necessary to start the interpolation:

Taks	Action
Send SYNC messages	see 6.5
Request operating mode	6060_h (modes_of_operation) = 7
Wait until operating mode is accepted	6061_h (modes_of_operation_display) = 7
Read current actual position	6064_h (position_actual_value)
Set read actual position as setpoint	$60C1_h$_01_h (ip_data_position)
Enable interpolation	6040_h (controlword), set `enable_ip_mode`
Wait for acknowledgement by servo drive	6041_h (statusword), query `ip_mode_active`
Move interpolated	

After completion of the synchronous movement, further evaluation of position values can be prevented by clearing the `enable_ip_mode` bit. If necessary, you can then switch to another operating mode.

If a running interpolation (`ip_mode_active` set) is interrupted by the occurrence of a servo drive error, the drive initially behaves as specified for the respective error (e.g. disabling the servo drive and change to `SWICTH_ON_DISABLED` state). Interpolation can then only be continued by resynchronization, since the servo drive must be returned to the `OPERATION_ENABLE` state, which clears the `ip_mode_active` bit.

5.4.3 Description of objects

5.4.3.1 Important objects in other sections

Index	Name	Section	Page
6040_h	controlword	*4 Device Control*	*102*
6041_h	statusword		
6093_h	position_factor	*3.3 Factor Group*	*43*
6094_h	velocity_encoder_factor		
6097_h	acceleration_factor		

5.4.3.2 Object $60C0_h$: interpolation_submode_select

The type of interpolation is defined via the interpolation_submode_select object. Currently, only the manufacturer-specific variant "Linear interpolation without buffer" is available.

Index	**$60C0_h$**			
Name	**interpolation_submode_select**			
Info	--	rw	PDO	INT16
Value	-2	--		

Value	Type of interpolation
-2	Linear interpolation without buffer

5.4.3.3 Object $60C1_h$: interpolation_data_record

The interpolation_data_record object represents the actual data set. It consists of an entry for the position value (ip_data_position) and a control word (ip_data_controlword), which specifies whether the position value is to be interpreted absolutely or relatively. The control word can be provided optionally. If it is not provided, the position value is interpreted as absolute. If the control word is also to be specified, subindex 2 (ip_data_controlword) must be written first and then subindex 1 (ip_data_position) for reasons of data consistency, since internally the data transfer is triggered with write access to ip_data_position.

Index	**$60C1_h$**			
Name	**interpolation_data_record**			
Type	RECORD			02_h
Sub-Index	**01_h**			
Name	**ip_data_position**			
Info	position_unit	rw	PDO	INT32
Value	--	--		
Sub-Index	**02_h**			
Name	**ip_data_controlword**			
Info	--	rw	PDO	UINT8
Value	0, 1	0		

Value	ip_data_position is
0	Absolute
1	Relative

INFORMATION **Internal data transfer**

The internal data transfer takes place with write access to sub-index 1. If sub-index 2 is also to be used, it must be written before sub-index 1.

5.4.3.4 Object $60C2_h$: interpolation_time_period

The synchronisation interval can be set via the interpolation_time_period object. The unit (ms or 1/10 ms) of the interval is defined via ip_time_unit and then set via ip_time_index. In Interpolated Position Mode the entire controller cascade (current, speed and position controller) is synchronised to the external clock. The change of the synchronisation interval therefore only becomes effective after a reset. If the interpolation interval is to be changed via the CAN bus, the parameter set must be saved (see section 3.1 *Loading and saving parameter sets* on page 38) and a reset must be executed (see section 6.6 *Network Management (NMT service)* on page 177) so that the new synchronisation interval takes effect.The synchronisation interval must be met exactly.

Index	**$60C2_h$**			
Name	**interpolation_time_period**			
Type	RECORD			02_h

Sub-Index	**01_h**			
Name	**ip_time_units**			
Info	according to ip_time_index	rw	PDO	UINT8
Value	ip_time_index = -3: 1, 2,..., 10 ip_time_index = -4: 10, 20,..., 100	--		

Sub-Index	**02_h**			
Name	**ip_time_index**			
Info	--	rw	PDO	INT8
Value	-3, -4	--		

Value	ip_time_index is given in
-3	10^{-3} seconds (ms)
-4	10^{-4} seconds (0.1 ms)

INFORMATION **Changing the synchronisation interval**

Changing the interpolation cycle time only takes effect after a reset. If the interpolation cycle time is to be changed via the CAN bus, the parameter set must be saved and a reset must be executed.

5.4.3.5 Object $60C3_h$: interpolation_sync_definition

Via the object interpolation_sync_definition the type (synchronize_on_group) and the number (ip_sync_every_n_event) of synchronisation telegrams per synchronisation interval is specified. Only the standard SYNC telegram and 1 SYNC per interval can be set.

Index	**$60C3_h$**			
Name	**interpolation_sync_definition**			
Type	ARRAY			02_h

Sub-Index	**01_h**			
Name	**syncronize_on_group**			
Info	--	rw	PDO	UINT8
Value	0	0		

Value	Description
0	Use standard SYNC telegram

Sub-Index	**02_h**			
Name	**ip_sync_every_n_event**			
Info	--	rw	PDO	UINT8
Value	1	1		

5.4.3.6 Object $60C4_h$: interpolation_data_configuration

The object record interpolation_data_configuration is intended for the configuration of an intermediate buffer. With the only available interpolation type "Linear interpolation without buffer" most entries have no meaning. However, even with this type of interpolation, access to object $60C1_h$ must be enabled via object buffer_clear!

Index	**$60C4_h$**	
Name	**interpolation_data_configuration**	
Type	RECORD	06_h

Sub-Index	**01_h**			
Name	**max_buffer_size**			
Info	--	ro	~~PDO~~	UINT32
Value	0	0		

Sub-Index	**02_h**			
Name	**actual_size**			
Info	--	rw	PDO	UINT32
Value	0	0		

Sub-Index	**03_h**			
Name	**buffer_organisation**			
Info	--	rw	PDO	UINT8
Value	0	0		

Value	Description
0	FIFO

Sub-Index	**04_h**			
Name	**buffer_position**			
Info	--	rw	PDO	UINT16
Value	0	0		

Sub-Index	**05_h**			
Name	**size_of_data_record**			
Info	--	wo	PDO	UINT8
Value	2	2		

Sub-Index	**06_h**			
Name	**buffer_clear**			
Info	--	wo	PDO	UINT8
Value	0, 1	0		

Value	Description
0	Delete Buffer / Access to $60C1_h$ not allowed
1	Access to $60C1_h$ released

5.4.3.7 Object 1006_h: communication_cycle_period

The set interpolation interval (=bus cycle time) can be read out via object 1006_h (communication_cycle_period). It is equal to the time t_P described in the section *Control circuit cycle times* in the Product manual BL 4000.

Index	**1006_h**			
Name	**communication_cycle_period**			
Info	µs	rw	~~PDO~~	UINT32
Value	--	00000000_h		

5.5 Cyclic Synchronous Position Mode

5.5.1 Overview

Just as in Interpolated Position Mode (IP), in Cyclic Synchronous Position Mode (CSP) the servo drive follows cyclic position setpoints in a multi-axis application of the servo drive.

The main differences are:

- The setpoint is specified via the target_position ($607A_h$)
- The setpoints are evaluated directly after changing to Cyclic Synchronous Position Mode. It is not necessary to set the bit `enable_ip_mode` in the controlword and also the object buffer_clear ($60C4_h$_06_h) must not be written.

5.5.2 Description of objects

5.5.2.1 Important objects in other sections

Index	Name	Section	Page
$607A_h$	target_position	*5.3.3.2 Object 607Ah: target_position*	136
$60C2_h$	interpolation_time_period	*5.4 Interpolated Position Mode*	102
6040_h	controlword	*4 Device Control*	102
6041_h	statusword		
6093_h	position_factor	*3.3 Factor Group*	43
6094_h	velocity_encoder_factor		
6097_h	acceleration_factor		

The Cyclic Synchronous Position Mode does not define its own objects.

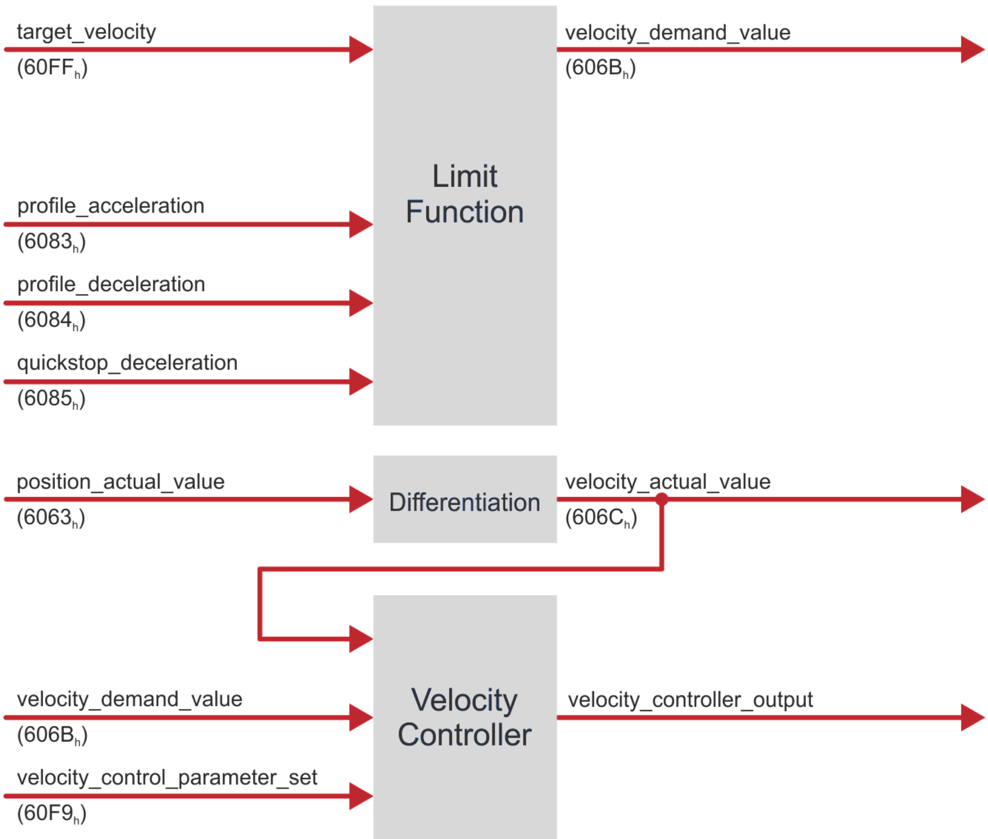
5.6 Profile Velocity Mode

5.6.1 Overview

The speed-controlled mode (Profile Velocity Mode) includes the following sub-functions:

- Setpoint generation by the ramp generator
- Speed control with suitable input and output signals
- Limitation of the torque setpoint (torque_demand_value)
- Monitoring of the actual velocity (velocity_actual_value) with the window function/threshold

The meaning of the following parameters is described in section 5.3 *Profile Position Mode* on page 134: profile_acceleration, profile_deceleration, quick_stop_deceleration.



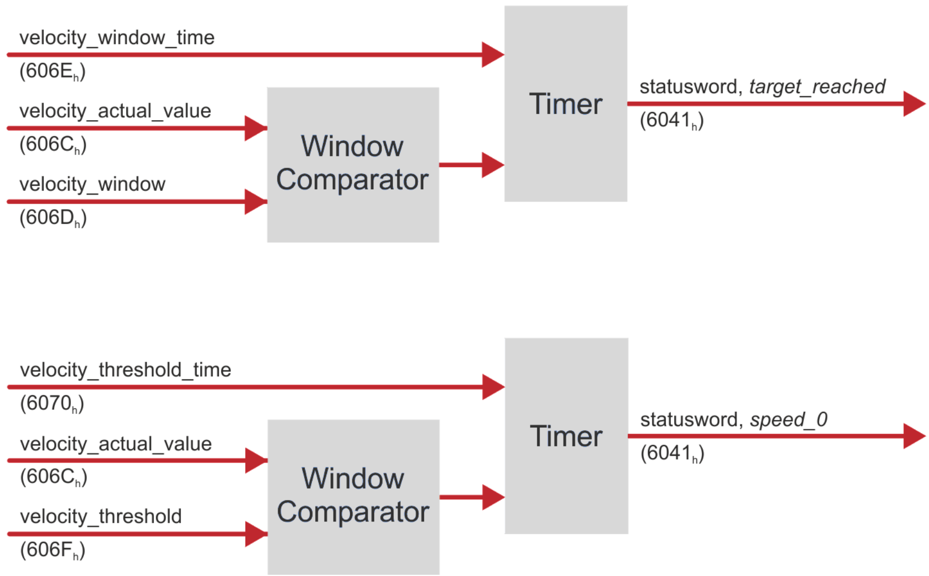


Figure 25: Structure of speed-controlled operation (Profile Velocity Mode)

5.6.2 Description of objects

5.6.2.1 Important objects in other sections

Index	Name	Sections	Page
6040_h	controlword	*4 Device Control*	*102*
6041_h	statusword		
6064_h	position_actual_value	*3.7 Position Controller*	*63*
6071_h	target_torque	*5.7 Profile Torque Mode*	*156*
6072_h	max_torque_value		
6083_h	profile_acceleration	*5.3 Profile Position Mode*	*134*
6084_h	profile_deceleration		
6085_h	quick_stop_deceleration		
6094_h	velocity_encoder_factor		

5.6.2.2 Object 6069_h: velocity_sensor_actual_value

With the object velocity_sensor_actual_value the value of a possible velocity encoder can be read out in internal units. No separate speed sensor can be connected to BL 4000 servo drives. Therefore, object $606C_h$ should always be used to determine the actual speed value.

Index	**6069_h**			
Name	**velocity_sensor_actual_value**			
Info	rev / 4096 min	ro	PDO	INT32
Value	--	--		

5.6.2.3 Object $606A_h$: sensor_selection_code

The speed sensor can be selected with this object. Currently, no separate speed sensor is provided. Therefore, only the standard angle encoder can be selected.

Index	**$606A_h$**			
Name	**sensor_selection_code**			
Info	--	rw	PDO	INT16
Value	0	0		

5.6.2.4 Object $606B_h$: velocity_demand_value

This object can be used to read out the current speed setpoint of the speed controller, which is generated by the ramp generator or the trajectory generator. If the position controller is activated, its correction speed is also added.

Index	**$606B_h$**			
Name	**velocity_demand_value**			
Info	speed_unit	ro	PDO	INT32
Value	--	--		

5.6.2.5 Object $202E_h$: velocity_demand_sync_value

The setpoint speed of the synchronisation encoder can be read out via this object. This is defined by object 2022_h synchronization_encoder_select (section 3.11 *Setpoint / actual value selection* on page 78).

Index	**$202E_h$**			
Name	**velocity_demand_sync_value**			
Info	speed_unit	ro	~~PDO~~	INT32
Value	--	--		

5.6.2.6 Object $606C_h$: velocity_actual_value

The actual speed value can be read out via this object.

Index	**$606C_h$**			
Name	**velocity_actual_value**			
Info	speed_unit	ro	PDO	INT32
Value	--	--		

5.6.2.7 Object 2074_h: velocity_actual_value_filtered

The velocity_actual_value_filtered object can be used to read out a filtered actual velocity value that should only be used for display purposes. In contrast to velocity_actual_value, velocity_actual_value_filtered is not used in the velocity control loop, but is used to protect the servo drive against overspeed. The filter time constant can be set via Object 2073_h (velocity_display_filter_time). See section 3.6.2.2 *Object 2073h: velocity_display_filter_time* on page 62.

Index	**2074_h**			
Name	**velocity_actual_value_filtered**			
Info	speed_unit	ro	PDO	INT32
Value		--		

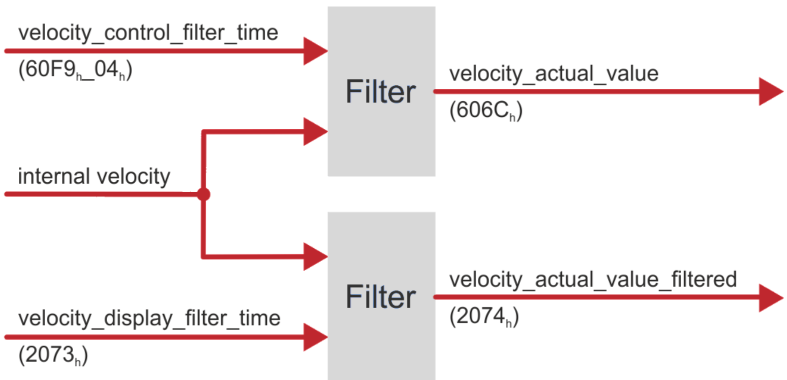


Figure 26: Determining velocity_actual_value and velocity_actual_value_filtered

5.6.2.8 Object 606D$_h$: velocity_window

The velocity_window_time and velocity_window objects are used to set the window comparator for comparing the actual speed value with the target velocity (object 60FF$_h$). To set bit 10 `target_reached` in the statusword object, the speed must be within velocity_window for the time specified in velocity_window_time.

Index	**606D$_h$**			
Name	**velocity_window**			
Info	speed_unit	rw	PDO	UINT16
Value	--	--		

5.6.2.9 Object 606E$_h$: velocity_window_time

The velocity_window_time and velocity_window objects are used to set the window comparator for comparing the actual speed value with the target velocity (object 60FF$_h$). To set bit 10 `target_reached` in the statusword object, the speed must be within velocity_window for the time specified in velocity_window_time.

Index	**606E$_h$**			
Name	**velocity_window_time**			
Info	ms	rw	PDO	UINT16
Value	0...4999	0		

5.6.2.10 Object 606F$_h$: velocity_threshold

The velocity_threshold and velocity_threshold_time objects specify the actual speed value at which the drive is considered to be standing still. If the drive exceeds the speed specified under velocity_threshold for velocity_threshold_time, bit 12 (velocity = 0) is deleted in the statusword.

Index	**606F$_h$**			
Name	**velocity_threshold**			
Info	speed_unit	rw	PDO	UINT16
Value	--	--		

5.6.2.11 Object 6070$_h$: velocity_threshold_time

The velocity_threshold and velocity_threshold_time objects specify the actual speed value at which the drive is considered to be standing still. If the drive exceeds the speed specified under velocity_threshold for velocity_threshold_time, bit 12 (velocity = 0) is deleted in the statusword.

Index	**6070$_h$**			
Name	**velocity_threshold_time**			
Info	ms	rw	PDO	UINT16
Value	0...4999	0		

5.6.2.12 Object 6080$_h$: max_motor_speed

The max_motor_speed object gives the highest permitted speed for the motor in min^{-1}. The object is used to protect the motor and can be taken from the motor data sheet. The speed setpoint is limited to this value.

Index	**6080$_h$**			
Name	**max_motor_speed**			
Info	min^{-1}	rw	PDO	UINT16
Value	0...32767	--		

5.6.2.13 Object 60FF$_h$: target_velocity

The target_velocity object is the setpoint for the ramp generator.

Index	**60FF$_h$**			
Name	**target_velocity**			
Info	speed_unit	rw	PDO	INT32
Value	--	--		

5.6.2.14 Speed ramps

If Profile Velocity Mode is selected as modes_of_operation, the setpoint ramp is also activated. Thus it is possible to limit a step-shaped setpoint change to a certain velocity change per time via the profile_acceleration and profile_deceleration objects. The servo drive not only offers the possibility to use different values for deceleration and acceleration, but also to set different accelerations for positive and negative speed. The following figure illustrates this behavior:

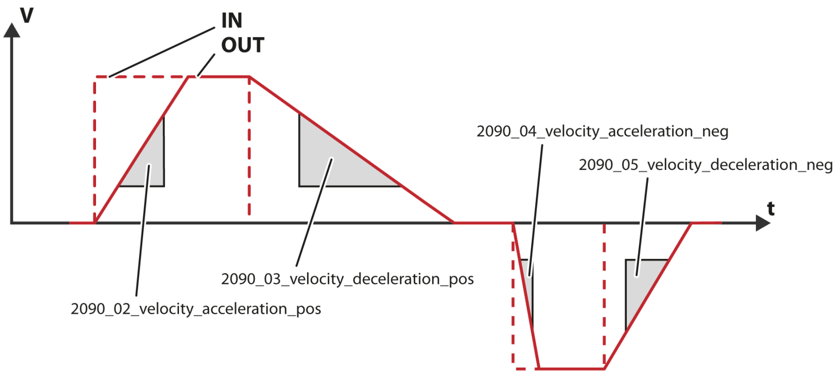


Figure 27: Speed ramps

To be able to parameterise these 4 accelerations individually, the velocity_ramps object group is available. It should be noted that the profile_acceleration and profile_deceleration objects change the same internal accelerations as the velocity_ramps. If the profile_acceleration is written, velocity_acceleration_pos and velocity_acceleration_neg are changed together, if the profile_deceleration is written, velocity_deceleration_pos and velocity_deceleration_neg are changed together.

If a 1 is written to the velocity_ramps_enable object, the setpoints are passed through the ramp generator.

Index	**2090_h**			
Name	**velocity_ramps**			
Type	RECORD			05_h

Sub-Index	**01_h**			
Name	**velocity_rampe_enable**			
Info	--	rw	~~PDO~~	UINT8
Value	0, 1	--		

Sub-Index	**02_h**			
Name	**velocity_acceleration_pos**			
Info	acceleration_unit	rw	~~PDO~~	INT32
Value	--	--		

Sub-Index	**03_h**			
Name	**velocity_deceleration_pos**			
Info	acceleration_unit	rw	~~PDO~~	INT32
Value	--	--		

Sub-Index	**04_h**			
Name	**velocity_acceleration_neg**			
Info	acceleration_unit	rw	~~PDO~~	INT32
Value	--	--		

Sub-Index	**05_h**			
Name	**velocity_deceleration_neg**			
Info	acceleration_unit	rw	~~PDO~~	INT32
Value	--	--		

5.7 Profile Torque Mode

5.7.1 Overview

This chapter describes the torque controlled operation. This operating mode allows the servo drive to use an external torque setpoint (target_torque), which can be smoothed by the integrated ramp generator. Thus it is possible to use the servo drive in applications where both the position controller and the speed controller are shifted to a superordinate control.

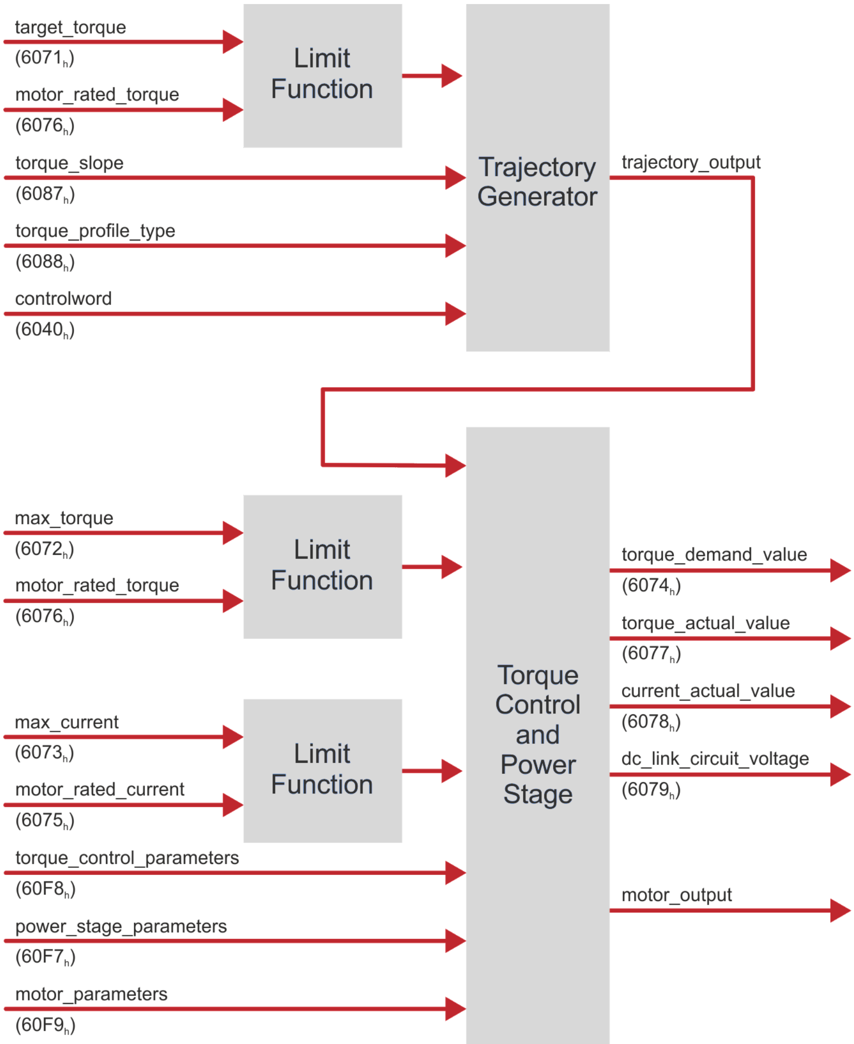


Figure 28: Structure of the torque-controlled operation mode

The torque_slope and torque_profile_type parameters must be specified for the ramp generator. If bit 8 `halt` is set in the controlword, the ramp generator reduces the torque to zero. Accordingly, it increases it again to the target_torque, if bit 8 is deleted again. In both cases the ramp generator considers the torque_slope and the torque_profile_type.

All definitions within this chapter refer to rotary motors. If linear motors are used, all "torque" objects must refer to a "force" instead. For simplicity, the objects are not duplicated and their names should not be changed. The operating modes Profile Position Mode and Profile Velocity Mode require the torque controller for their function. Therefore, it is always necessary to parameterise it.

5.7.2 Description of objects

5.7.2.1 Important objects from other sections

Index	Name	Section	Page
6040_h	controlword	*4 Device Control*	*92*
$60F9_h$	motor_parameters	*3.5 Current controller and motor adaption*	*54*
6075_h	motor_rated_current		
6073_h	max_current		

5.7.2.2 Object 6071_h: target_torque

This parameter is the input value for the torque controller in torque-controlled mode (section 5.7 *Profile Torque Mode* on page 156). It is specified in thousandths of the nominal torque (object 6076_h).

Index	**6071_h**			
Name	**target_torque**			
Info	‰ (1000 = motor_rated torque)	rw	PDO	INT16
Value	--	--		

5.7.2.3 Object 6072_h: max_torque

This value represents the maximum permissible torque of the motor. It is specified in thousandths of the nominal torque (object 6076_h). If, for example, a twofold overload of the motor is permissible for a short time, the value 2000 must be entered here.

INFORMATION **Object 6072_h and Object 6073_h are dependent on each other**

Object 6072_h (max_torque) and object 6073_h (max_current) are dependent on each other and may only be written if object 6075_h(motor_rated_current) has been written with a valid value beforehand.

Index	**6072_h**			
Name	**max_torque**			
Info	‰ (1000 = motor_rated torque)	rw	PDO	UINT16
Value	1000...65535	--		

5.7.2.4 Object 6074_h: torque_demand_value

This object can be used to read out the current torque setpoint in thousandths of the nominal torque (6076_h). The internal limitations of the servo drive (current limits and I^2t-monitoring) are taken into account here.

Index	**6074_h**			
Name	**torque_demand_value**			
Info	‰ (1000 = motor_rated torque)	ro	PDO	INT16
Value	--	--		

5.7.2.5 Object 6076_h: motor_rated_torque

This object indicates the nominal torque of the motor. This can be taken from the type plate of the motor. It must be entered in the unit 0.001 Nm.

Index	**6076_h**			
Name	**motor_rated_torque**			
Info	0.001 Nm	rw	PDO	UINT32
Value	--	--		

5.7.2.6 Object 6077_h: torque_actual_value

This object can be used to read out the actual torque value of the motor in thousandths of the nominal torque (object 6076_h).

Index	**6077_h**			
Name	**torque_actual_value**			
Info	‰ (1000 = motor_rated torque)	ro	PDO	INT16
Value	--	--		

5.7.2.7 Object 6078_h: current_actual_value

This object can be used to read out the actual current value of the motor in thousandths of the rated current (object 6075_h).

Index	**6078_h**			
Name	**current_actual_value**			
Info	‰ (1000 = motor_rated current)	ro	PDO	INT16
Value	--	--		

5.7.2.8 Object 6079h: dc_link_circuit_voltage

The DC link voltage of the servo drive can be read out via this object. The voltage is specified in the unit millivolts.

Index	**6079_h**			
Name	**dc_link_circuit_voltage**			
Info	mV	ro	PDO	UINT32
Value	--	--		

5.7.2.9 Object 6087_h: torque_slope

This parameter describes the rate of change of the setpoint ramp. This is to be specified in thousandths of the nominal torque per second. For example, the torque setpoint target_torque is increased from 0 Nm to the value motor_rated_torque. If the output value of the torque ramp should reach this value in one second, then the value 1000 must be written into this object.

Index	**6087_h**			
Name	**torque_slope**			
Info	motor_rated_torque / 1000 s	rw	PDO	UINT32
Value	--	--		

5.7.2.10 Object 6088_h: torque_profile_type

The torque_profile_type object is used to specify the waveform with which a setpoint step is executed. At present, only the linear ramp is implemented in this servo drive, so that this object can only be written with the value 0.

Index	**6088_h**			
Name	**torque_profile_type**			
Info	--	rw	PDO	INT16
Value	0	0		

Value	Description
0	Linear ramp

6 Detailed description of the CANopen protocol

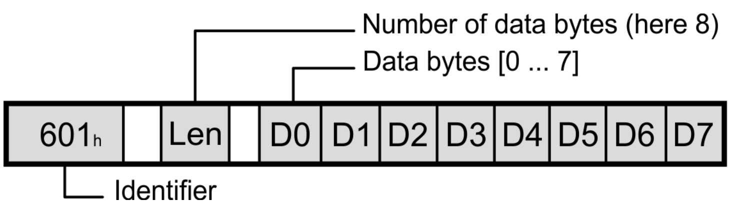
6.1 Introduction

CANopen provides a simple and standardised way to access the parameters of the servo drive (e.g. the maximum motor current). For this purpose, each parameter (CAN object) is assigned a unique number (index and subindex). The totality of all parameters is called the object dictionary. Two main methods are available for accessing the CAN objects via the CAN bus: A confirmed access method, in which the servo drive acknowledges each parameter access (via SDOs) and an unconfirmed access method, in which no acknowledgement is made (via PDOs). As a rule, the servo drive is parameterised via SDOs, while the cyclic process data is exchanged via PDOs.

The following communication objects are defined in total:

SDO	Service Data Object	Are used for normal parameterization of the servo drive
PDO	Process Data Object	Fast exchange of process data (e.g. actual speed) possible
SYNC	Synchronization Message	Synchronization of multiple CAN nodes
EMCY	Emergency Message	Transfer of error messages
NMT	Network Management	Network service: For example, all CAN nodes can be acted upon simultaneously
BOOTUP	Error Control Protocol	Bootup message
HEARTBEAT	Error Control Protocol	Monitoring of communication participants through periodic messages
NODEGUARDING	Error Control Protocol	Monitoring of communication participants through periodic messages

Each message sent on the CAN bus contains a type of address which can be used to determine for which bus station the message is intended. This number is called identifier. The lower the identifier, the higher the priority of the message. Identifiers are defined for each of the communication objects mentioned above. The following figure shows the basic structure of a CANopen message:



6.2 Access via SDO

The object dictionary of the servo drive can be accessed via the **s**ervice **d**ata **o**bjects (SDO). SDO accesses always originate from the superordinate control (host). The host sends either a write command to the servo drive to change a parameter of the object dictionary or a read command to read out a parameter. For each command, the host receives a response that either contains the read value or - in the case of a write command - serves as an acknowledgement. To allow the servo drive to recognise that the command is intended for it, the host must send the command with a specific identifier. **This identifier consists of the base 600h + node number of the servo drive concerned. The servo drive responds accordingly with the identifier 580h + node number.** The structure of the commands or the responses depends on the data type of the object to be read or written, since either 1, 2 or 4 data bytes must be sent or received. The following data types are supported:

UINT8	8-bit value unsigned	0 ... 255
INT8	8-bit value signed	-128 ... 127
UINT16	16-bit value unsigned	0 ... 65536
INT16	16-bit value signed	-32768 ... 32767
UINT32	32-bit value unsigned	0 ... $(2^{32} - 1)$
INT32	32-bit value signed	$-(2^{31})$ ... $(2^{31} - 1)$
VISSTR	Visible string	---

6.2.1 SDO sequences for reading and writing

In order to read or write objects of these number types, the sequences listed below are to be used. The commands for writing a value to the servo drive start with a different identifier depending on the data type. The response identifier, however, is always the same. Read commands always start with the same identifier and the servo drive responds differently depending on the data type returned. All numbers are in hexadecimal notation.

IX0: Low byte of the main index (hex)
IX1: High byte of the main index (hex)
SU: Subindex (hex)

UINT8 / INT8

	Read commands	Write commands
Command:	40_h IX0 IX1 SU	$2F_h$ IX0 IX1 SU D0
Response:	$4F_h$ IX0 IX1 SU D0	60_h IX0 IX1 SU

Read: Identifier for 8 bit ($4F_h$). Write: Identifier for 8 bit ($2F_h$).

UINT16 / INT16

	Read commands	Write commands
Command:	40_h IX0 IX1 SU	$2B_h$ IX0 IX1 SU D0 D1
Response:	$4B_h$ IX0 IX1 SU D0 D1	60_h IX0 IX1 SU

Read: Identifier for 16 bit ($4B_h$). Write: Identifier for 16 bit ($2B_h$).

UINT32 / INT32

	Read commands	Write commands
Command:	40_h IX0 IX1 SU	23_h IX0 IX1 SU D0 D1 D2 D3
Response:	43_h IX0 IX1 SU D0 D1 D2 D3	60_h IX0 IX1 SU

Read: Identifier for 32 bit (43_h). Write: Identifier for 32 bit (23_h).

EXAMPLE

UINT8 / INT8

Reading object 6061_h_00_h, Return data: 01_h — Writing object 1401_h_02_h, Data: EF_h

	Read	Write
Command:	40_h 61_h 60_h 00_h	$2F_h$ 01_h 14_h 02_h EF_h
Response:	$4F_h$ 61_h 60_h 00_h 01_h	60_h 01_h 14_h 02_h

UINT16 / INT16

Reading object 6041_h_00_h, Return data: 1234_h — Writing object 6040_h_00_h, Data: $03E8_h$

	Read	Write
Command:	40_h 41_h 60_h 00_h	$2B_h$ 40_h 60_h 00_h $E8_h$ 03_h
Response:	$4B_h$ 41_h 60_h 00_h 34_h 12_h	60_h 40_h 60_h 00_h

UINT32 / INT32

Reading object 6093_h_01_h, Return data: 12345678_h — Writing object 6093_h_01_h, Data: 12345678_h

	Read	Write
Command:	40_h 93_h 60_h 01_h	23_h 93_h 60_h 01_h 78_h 56_h 34_h 12_h
Response:	43_h 93_h 60_h 01_h 78_h 56_h 34_h 12_h	60_h 93_h 60_h 01_h

INFORMATION **Wait for the acknowledgement from the servo drive!**

Only when the servo drive has acknowledged the request further requests may be sent.

6.2.2 SDO error response (abort codes)

In case of an error during reading or writing (e.g. because the written value is too large), the servo drive responds with an error code instead of acknowledging:

Command:	...	IX0	IX1	SU	...	...	...	...	
Response:	43_h	IX0	IX1	SU	F0	F1	F2	F3	
	↑ Error identifier				↑	↑	↑	↑	Error code (4 byte)

Error code F3 F2 F1 F0	Meaning
$05\ 03\ 00\ 00_h$	Toggle bit not alternated
$05\ 04\ 00\ 01_h$	Client/server command specifier not valid or unknown
$06\ 01\ 00\ 00_h$	Unsupported access to an object
$06\ 01\ 00\ 01_h$	Attempt to read a write only object
$06\ 01\ 00\ 02_h$	Attempt to write a read only object
$06\ 02\ 00\ 00_h$	Object does not exist in the object dictionary
$06\ 04\ 00\ 41_h$	Object cannot be mapped to the PDO (e.g. a ro-object in an RPDO)
$06\ 04\ 00\ 42_h$	The number and length of the objects to be mapped would exceed PDO length
$06\ 04\ 00\ 43_h$	General parameter incompatibility reason
$06\ 04\ 00\ 47_h$	General internal incompatibility in the device
$06\ 06\ 00\ 00_h$	Access failed due to an hardware error *1)
$06\ 07\ 00\ 10_h$	Data type does not match, length of service parameter does not match
$06\ 07\ 00\ 12_h$	Data type does not match, length of service parameter too high
$06\ 07\ 00\ 13_h$	Data type does not match, length of service parameter too low
$06\ 09\ 00\ 11_h$	Sub-index does not exist
$06\ 09\ 00\ 30_h$	Value range of parameter exceeded (only for write access)
$06\ 09\ 00\ 31_h$	Value of parameter written too high
$06\ 09\ 00\ 32_h$	Value of parameter written too low
$06\ 09\ 00\ 36_h$	Maximum value is less than minimum value
$08\ 00\ 00\ 20_h$	Data cannot be transferred or stored to the application *1)
$08\ 00\ 00\ 21_h$	Data cannot be transferred or stored to the application because of local control
$08\ 00\ 00\ 22_h$	Data cannot be transferred or stored to the application because of the present device state *3)
$08\ 00\ 00\ 23_h$	Object dictionary dynamic generation fails or no object dictionary is present *2)

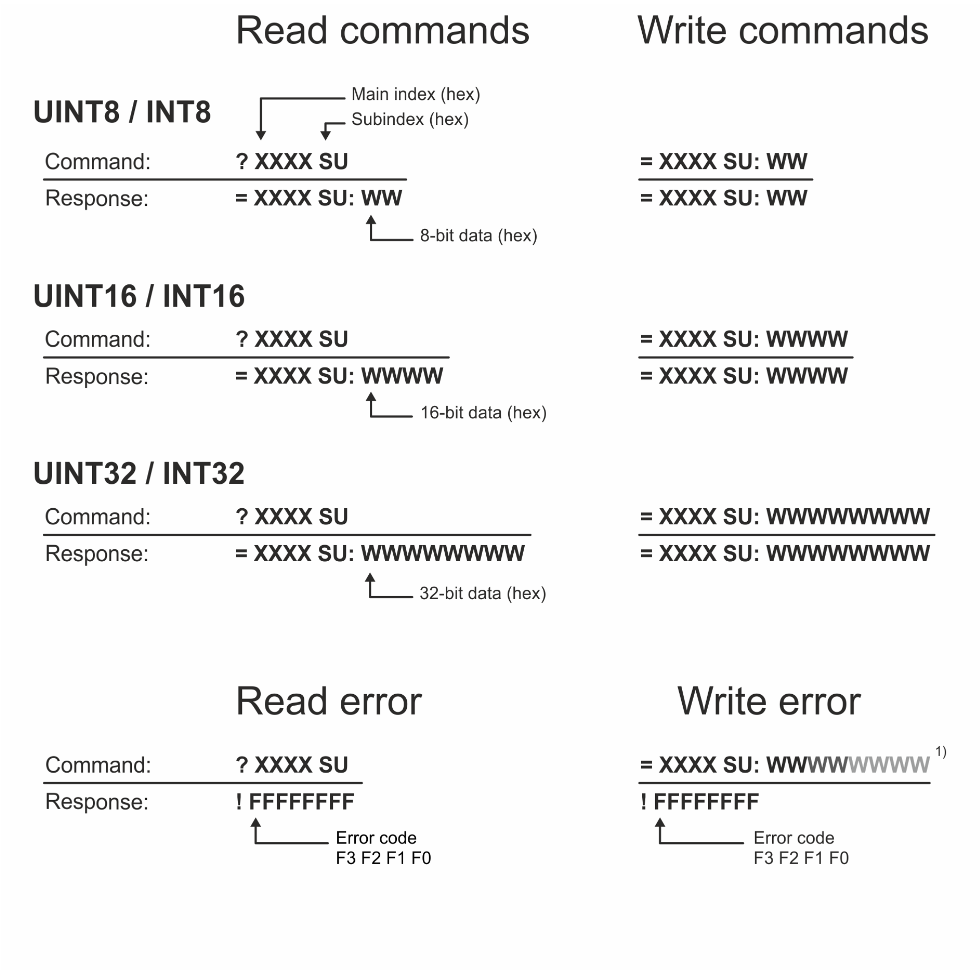
*1) This abort code is returned according to DS301 if store_parameters / restore_parameters are accessed incorrectly.

*2) This abort code is returned e.g. if another bus system controls the servo drive or parameter access is not allowed.

*3) "Device state" is to be understood generally here: It can be the wrong operating mode, as well as a non-existent technology module or similar.

6.2.3 Simulation of SDO accesses

The firmware of the servo drives offers the possibility to simulate SDO accesses via the parameterisation interface (e.g. the transfer window of the MPC ServoCommander). Thus, objects written via the CAN bus can be read and controlled via the parameterization interface. The syntax of the commands is:



1) In the case of an error, the response is structured the same for all 3 write commands (8, 16, 32 bits). The commands are entered as characters without any spaces.

INFORMATION **Test commands are not real-time capable**

Access via the parameterisation interface is not suitable for real-time communication.

6.3 Access via PDO

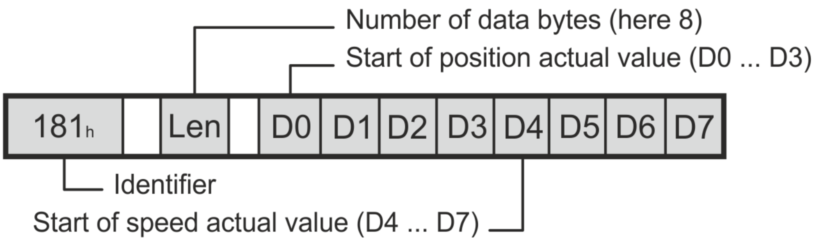
Process **d**ata **o**bjects (PDOs) can be used to transfer data in an event-controlled manner. The PDO only transfers payload data. Which parameters are transferred is defined in advance between host and servo. In contrast to an SDO, there is no acknowledgement when a PDO is transmitted. The following types of PDOs are distinguished:

Transmit-PDO (TPDO)	Servo→ Host	Servo drive sends PDO on occurrence of a specific event
Receive-PDO (RPDO)	Host→ Servo	Servo drive evaluates PDO on occurrence of a specific event

The servo drive has four transmit and four receive PDOs.

Almost all objects of the object dictionary can be mapped into the PDOs, for example the actual speed value, the actual position value or similar.

In the example below, the position actual value would be transmitted in data bytes 0...3 of the PDO and the speed actual value in bytes 4...7.



In this way almost any data telegrams can be defined. The following chapters describe the settings required for this.

6.3.1 Description of objects

› Identifier of the PDO

COB_ID_used_by_PDO

The identifier on which the respective PDO is to be sent or received must be entered in the object COB_ID_used_by_PDO. If bit 31 is set, the respective PDO is deactivated. This is the default setting for all PDOs. The COB-ID may only be changed if the PDO is deactivated, i.e. bit 31 is set. An identifier other than currently set in the servo drive may therefore only be written if bit 31 is set at the same time. The set bit 30 when reading the identifier indicates that the object cannot be queried by a remote frame. This bit is ignored on writing and is always set on reading.

› Number of objects to be transferred

number_of_mapped_objects

This object specifies how many objects are to be mapped into the corresponding PDO. The following restrictions must be observed:

- A maximum of 4 objects can be mapped per PDO
- A PDO may have a maximum of 64 bits (8 bytes)

› Objects to be transferred

first_mapped_object ... fourth_mapped_object

For each object to be contained in the PDO, the servo drive must be given the corresponding index, subindex and length. The length specification must match the length specification in the Object Dictionary. Parts of an object cannot be mapped.

The mapping entry is composed as follows:

Index (16 Bit), Subindex (8 Bit), Length (8 Bit)

To simplify the mapping, the following procedure is prescribed:

1. The number of mapped objects must be set to 0.
2. The parameters first_mapped_object...fourth_mapped_object may be written (The total length of all objects is not relevant at this time).
3. The number of mapped objects is set to a value between 1...4. The length of all these objects must not exceed 64 bits now.

Transmission type

transmission_type and inhibit_time

For each PDO it can be defined which event causes the transmission (Transmit-PDO) or the evaluation (Receive-PDO) of a message:

Value	Meaning	allowed with
01_h –$F0_h$	**SYNC message** The numerical value indicates how many SYNC messages must have arrived before the PDO is • sent (T-PDO) or • evaluated (R-PDO)	TPDOs RPDOs
FE_h	**Cyclic** The Transmit PDO is updated and sent cyclically by the servo drive. The time period is defined by the object inhibit_time in 100µs steps. In contrast, receive PDOs are evaluated immediately after receipt.	TPDOs (RPDOs)
FF_h	**Change** The Transmit PDO is sent when at least 1 bit has changed in the data of the PDO. This transmission_type is also permitted for Receive-PDOs. In addition, the inhibit_time can be used to define the minimum interval between the transmission of two PDOs in 100µs steps.	TPDOs

Masking

transmit_mask_high and transmit_mask_low

If "Change" is selected as transmission_type, the TPDO is always sent if at least 1 bit of the TPDO changes. But often it is needed that the TPDO is only sent if certain bits have changed. Therefore the TPDO can be provided with a mask: Only the bits of the TPDO that are set to "1" in the mask are used to evaluate whether the PDO has changed. Since this function is manufacturer specific, all bits of the masks are set by default.

EXAMPLE

The following objects are to be transmitted together in one PDO:

Index_Subindex	Length	Object name
6041_h_00_h	10_h	statusword
6061_h_00_h	08_h	modes_of_operation_display
$60FD_h$_00_h	20_h	digital_inputs

The first Transmit PDO (TPDO 1) is to be used, which is to be sent whenever one of the digital inputs changes, but at most every 10 ms. 187h is to be used as identifier for this PDO.

1. **Deactivate PDO**
   If the PDO is active, it must first be deactivated, i.e. the identifier must be written with bit 31 set:
   `cob_id_used_by_pdo = C0000187h`

2. **Delete number of objects**
   To allow changing the object mapping, the number of mapped objects must be set to zero.
   `number_of_mapped_objects = 0`

3. **Configuring objects**
   Index and subindex of the objects listed above must each be combined to a 32 bit value.
   `first_mapped_object =` 60410010_h
   `second_mapped_object =` 60610008_h
   `third_mapped_object =` $60FD0020_h$

4. **Set number of objects**
   Three objects are to be transmitted in the PDO.
   `number_of_mapped_objects = 3`

5. **Set transmission type**
   The PDO should be sent on change of the digital inputs. To ensure that only the change of the digital inputs leads to sending, the PDO is masked.The PDO should be sent at most every 10 ms (100x100µs).
   `transmission_type =` FF_h
   `transmit_mask_low =` $000000FF_h$
   `transmit_mask_high =` $FFFFFF00_h$
   `inhibit_time =` 64_h

6. **Set identifier**
   The PDO is to be sent with identifier 187h: Writing of the identifier with deleted bit 31:
   `cob_id_used_by_pdo =` 40000187_h

INFORMATION **Changing the PDO settings**

Note that the PDO configuration may generally only be changed if the network status (NMT) is not Operational. See also section 6.6 *Network Management (NMT service)* on page 177.

6.3.2 Objects for PDO configuration

The individual objects for configuring the PDOs are the same for all 4 TPDOs and all 4 RPDOs. Therefore only the parameter description of the first TPDO is explicitly listed below. It is to be used analogously also for the other PDOs, which are listed tabularly in the following:

Index	**1800_h**			
Name	**transmit_pdo_parameter_tpdo1**			
Type	RECORD			03_h
Sub-Index	**01_h**			
Name	**cob_id_used_by_pdo_tpdo1**			
Info	--	rw	~~PDO~~	UINT32
Value	181_h...$1FF_h$, Bit 30 and 31 may be set	$C0000181_h$		
Sub-Index	**02_h**			
Name	**transmission_type_tpdo1**			
Info	--	rw	~~PDO~~	UINT8
Value	0...$8C_h$, FE_h, FF_h	FF_h		
Sub-Index	**03_h**			
Name	**inhibit_time_tpdo1**			
Info	100µs (10 = 1ms)	rw	~~PDO~~	UINT16
Value	--	0000_h		

Index	**$1A00_h$**			
Name	**transmit_pdo_mapping_tpdo1**			
Type	RECORD			04_h
Sub-Index	**00_h**			
Name	**number_of_mapped_objects_tpdo1**			
Info	--	rw	~~PDO~~	UINT8
Value	0...4	see Table		
Sub-Index	**01_h**			
Name	**first_mapped_object_tpdo1**			
Info	--	rw	~~PDO~~	UINT32
Value	--	see Table		
Sub-Index	**02_h**			
Name	**second_mapped_object_tpdo1**			
Info	--	rw	~~PDO~~	UINT32
Value	--	see Table		
Sub-Index	**03_h**			
Name	**third_mapped_object_tpdo1**			
Info	--	rw	~~PDO~~	UINT32
Value	--	see Table		

Sub-Index	04_h			
Name	**fourth_mapped_object_tpdo1**			
Info	--	rw	~~PDO~~	UINT32
Value	--	see Table		

INFORMATION **PDO must be deactivated before configuring.**

Note that the object groups transmit _pdo_parameter_xxx and transmit_pdo_mapping_xxx can only be written if the PDO is deactivated (Bit 31 in cob_id_used_by_pdo_xxx set).

1. Transmit PDO

Index	Comment	Type	Acc.	Default Value
1800_h_00_h	number of entries	UINT8	ro	03_h
1800_h_01_h	COB-ID used by PDO	UINT32	rw	$C0000181_h$
1800_h_02_h	transmission type	UINT8	rw	FF_h
1800_h_03_h	inhibit time (100 µs)	UINT16	rw	0000_h
$1A00_h$_00_h	number of mapped objects	UINT8	rw	01_h
$1A00_h$_01_h	first mapped object	UINT32	rw	60410010_h
$1A00_h$_02_h	second mapped object	UINT32	rw	00000000_h
$1A00_h$_04_h	fourth mapped object	UINT32	rw	00000000_h

tpdo_1_transmit_mask

Index	Comment	Type	Acc.	Default Value
2014_h_00_h	number of entries	UINT8	ro	02h
2014_h_01_h	tpdo_1_transmit_mask_low	UINT32	rw	FFFFFFFFh
2014_h_02_h	tpdo_1_transmit_mask_high	UINT32	rw	FFFFFFFFh

2. Transmit PDO

Index	Comment	Type	Acc.	Default Value
1801_h_00_h	number of entries	UINT8	ro	03_h
1801_h_01_h	COB-ID used by PDO	UINT32	rw	$C0000281_h$
1801_h_02_h	transmission type	UINT8	rw	FF_h
1801_h_03_h	inhibit time (100 µs)	UINT16	rw	0000_h
$1A01_h$_00_h	number of mapped objects	UINT8	rw	02_h
$1A01_h$_01_h	first mapped object	UINT32	rw	60410010_h
$1A01_h$_02_h	second mapped object	UINT32	rw	60610008_h
$1A01_h$_03_h	third mapped object	UINT32	rw	00000000_h
$1A01_h$_04_h	fourth mapped object	UINT32	rw	00000000_h

tpdo_2_transmit_mask

Index	Comment	Type	Acc.	Default Value
$2015_h\_00_h$	number of entries	UINT8	ro	02h
$2015_h\_01_h$	tpdo_2_transmit_mask_low	UINT32	rw	FFFFFFFFh
$2015_h\_02_h$	tpdo_2_transmit_mask_high	UINT32	rw	FFFFFFFFh

3. Transmit PDO

Index	Comment	Type	Acc.	Default Value
$1802_h\_00_h$	number of entries	UINT8	ro	03_h
$1802_h\_01_h$	COB-ID used by PDO	UINT32	rw	$C0000381_h$
$1802_h\_02_h$	transmission type	UINT8	rw	FF_h
$1802_h\_03_h$	inhibit time (100 µs)	UINT16	rw	0000_h
$1A02_h\_00_h$	number of mapped objects	UINT8	rw	02_h
$1A02_h\_01_h$	first mapped object	UINT32	rw	60410010_h
$1A02_h\_02_h$	second mapped object	UINT32	rw	60640020_h
$1A02_h\_03_h$	third mapped object	UINT32	rw	00000000_h
$1A02_h\_04_h$	fourth mapped object	UINT32	rw	00000000_h

tpdo_3_transmit_mask

Index	Comment	Type	Acc.	Default Value
$2016_h\_00_h$	number of entries	UINT8	ro	02h
$2016_h\_01_h$	tpdo_3_transmit_mask_low	UINT32	rw	FFFFFFFFh
$2016_h\_02_h$	tpdo_3_transmit_mask_high	UINT32	rw	FFFFFFFFh

4.Transmit PDO

Index	Comment	Type	Acc.	Default Value
$1803_h\_00_h$	number of entries	UINT8	ro	03_h
$1803_h\_01_h$	COB-ID used by PDO	UINT32	rw	$C0000481_h$
$1803_h\_02_h$	transmission type	UINT8	rw	FF_h
$1803_h\_03_h$	inhibit time (100 µs)	UINT16	rw	0000_h
$1A03_h\_00_h$	number of mapped objects	UINT8	rw	02_h
$1A03_h\_01_h$	first mapped object	UINT32	rw	60410010_h
$1A03_h\_02_h$	second mapped object	UINT32	rw	$606C0020_h$
$1A03_h\_03_h$	third mapped object	UINT32	rw	00000000_h
$1A03_h\_04_h$	fourth mapped object	UINT32	rw	00000000_h

tpdo_4_transmit_mask

Index	Comment	Type	Acc.	Default Value
$2017_h\_00_h$	number of entries	UINT8	ro	02h
$2017_h\_01_h$	tpdo_4_transmit_mask_low	UINT32	rw	FFFFFFFFh
$2017_h\_02_h$	tpdo_4_transmit_mask_high	UINT32	rw	FFFFFFFFh

1. Receive PDO

Index	Comment	Type	Acc.	Default Value
$1400_h\_00_h$	number of entries	UINT8	ro	02_h
$1400_h\_01_h$	COB-ID used by PDO	UINT32	rw	$C0000201_h$
$1400_h\_02_h$	transmission type	UINT8	rw	FF_h
$1600_h\_00_h$	number of mapped objects	UINT8	rw	01_h
$1600_h\_01_h$	first mapped object	UINT32	rw	60400010_h
$1600_h\_02_h$	second mapped object	UINT32	rw	00000000_h
$1600_h\_03_h$	third mapped object	UINT32	rw	00000000_h
$1600_h\_04_h$	fourth mapped object	UINT32	rw	00000000_h

2. Receive PDO

Index	Comment	Type	Acc.	Default Value
$1401_h\_00_h$	number of entries	UINT8	ro	02_h
$1401_h\_01_h$	COB-ID used by PDO	UINT32	rw	$C0000301_h$
$1401_h\_02_h$	transmission type	UINT8	rw	FF_h
$1601_h\_00_h$	number of mapped objects	UINT8	rw	02_h
$1601_h\_01_h$	first mapped object	UINT32	rw	60400010_h
$1601_h\_02_h$	second mapped object	UINT32	rw	60600008_h
$1601_h\_03_h$	third mapped object	UINT32	rw	00000000_h
$1601_h\_04_h$	fourth mapped object	UINT32	rw	00000000_h

3. Receive PDO

Index	Comment	Type	Acc.	Default Value
$1402_h\_00_h$	number of entries	UINT8	ro	02_h
$1402_h\_01_h$	COB-ID used by PDO	UINT32	rw	$C0000401_h$
$1402_h\_02_h$	transmission type	UINT8	rw	FF_h
$1602_h\_00_h$	number of mapped objects	UINT8	rw	02_h
$1602_h\_01_h$	first mapped object	UINT32	rw	60400010_h
$1602_h\_02_h$	second mapped object	UINT32	rw	$607A0020_h$
$1602_h\_03_h$	third mapped object	UINT32	rw	00000000_h
$1602_h\_04_h$	fourth mapped object	UINT32	rw	00000000_h

4. Receive PDO

Index	Comment	Type	Acc.	Default Value
$1403_h\_00_h$	number of entries	UINT8	ro	02_h
$1403_h\_01_h$	COB-ID used by PDO	UINT32	rw	$C0000501_h$
$1403_h\_02_h$	transmission type	UINT8	rw	FF_h
$1603_h\_00_h$	number of mapped objects	UINT8	rw	02_h
$1603_h\_01_h$	first mapped object	UINT32	rw	60400010_h
$1603_h\_02_h$	second mapped object	UINT32	rw	$60FF0020_h$
$1603_h\_03_h$	third mapped object	UINT32	rw	00000000_h
$1603_h\_04_h$	fourth mapped object	UINT32	rw	00000000_h

6.3.3 Activation of PDOs

The following points must be fulfilled for the servo drive to **send** or **evaluate** PDOs:

- The object number_of_mapped_objects must be non-zero.
- Bit 31 in the cob_id_used_for_pdos object must be cleared.
- The communication status of the servo drive must be Operational (see section 6.6 *Network Management (NMT service)* on page 177)

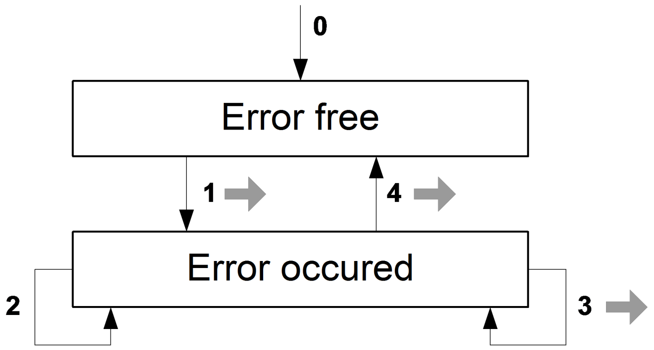
The communication status of the servo drive must not be Operational so that PDOs can be **configured**.

6.4 EMERGENCY message

The servo drive monitors the function of its main assemblies. These include the power supply, the power stage, the angle encoder evaluation and the technology slots available on some servo drives. In addition, the motor (temperature, angle encoder) and the limit switches are continuously monitored. Incorrect parameterizations can also lead to error messages (division by zero, etc.).

6.4.1 Overview

The servo drive sends an EMERGENCY message when an error occurs or when an error is acknowledged. The identifier of this message is composed of the identifier 80h and the node number of the servo drive concerned.

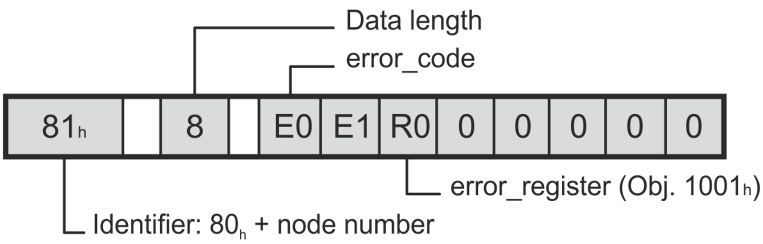


After a reset, the servo drive is in the Error free state (which it may leave again immediately if an error has been present from the start). The following state transitions are possible:

Nr.	Cause	Description
0	Initialization completed	
1	Error occurs	There was no error and an error occurs. An EMERGENCY telegram with the error code of the occurred error is sent
2	Error acknowledgement	An error acknowledgement (see section 4.3 *controlword* on page 108) is attempted, but not all causes are resolved.
3	Error occurs	There is already an error and another error occurs. An EMERGENCY telegram with the error code of the new error is sent.
4	Error acknowledgement	An error acknowledgement is attempted and all causes are eliminated. An EMERGENCY telegram with error code 0000 is sent.

6.4.2 Structure of the EMERGENCY message

The EMERGENCY message consists of eight data bytes, where the first two bytes contain an error_code. The third byte is intended for another error code (object 1001_h), but does not contain any relevant information. The remaining five bytes contain zeros.



An overview of all error codes that may occur can be found in section 7.3 *Error codes of the EMERGENCY message* on page 186

6.4.3 Description of objects

Object 1003_h: pre_defined_error_field

The respective error_code of the error messages is additionally stored in a four-level error memory. This is structured like a shift register so that the last error that occurred is always stored in object 1003_h_01_h (standard_error_field_0). By a read access to the object 1003_h_00_h (pre_defined_error_field) it can be determined how many error messages are currently stored in the error memory. The error memory is cleared by writing the value 0 into the object 1003_h_00_h (pre_defined_error_field). In order to be able to reactivate the output stage of the servo drive after an error, an error acknowledgement (`reset_fault`, see section 4.3 *controlword* on page 108) must also be performed.

Index	**1003_h**			
Name	**pre_defined_error_field**			
Type	ARRAY			04_h

Sub-Index	**01_h**			
Name	**standard_error_field_0**			
Info	--	ro	~~PDO~~	UINT32
Value	--	00000000_h		

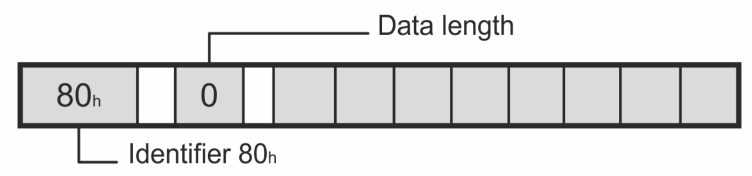
Sub-Index	**02_h**			
Name	**standard_error_field_1**			
Info	--	ro	~~PDO~~	UINT32
Value	--	00000000_h		

Sub-Index	**03_h**			
Name	**standard_error_field_2**			
Info	--	ro	~~PDO~~	UINT32
Value	--	00000000_h		

Sub-Index	**04_h**			
Name	**standard_error_field_3**			
Info	--	ro	~~PDO~~	UINT32
Value	--	00000000_h		

6.5 SYNC message

Several devices of a plant can be synchronised with each other. For this purpose, one of the devices (usually the superordinate control) periodically sends out synchronization messages. All connected servo drives receive these messages and use them to handle the PDOs (siehe section 6.3 *Access via PDO* on page 165).



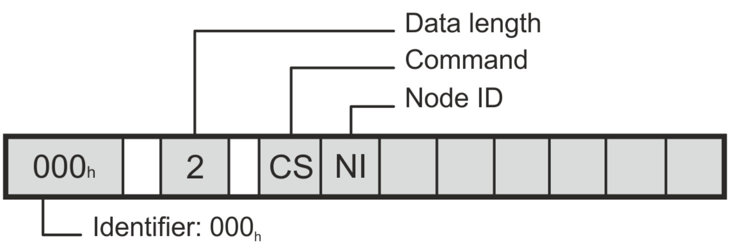
The identifier on which the servo drive receives the SYNC message is fixed at 80_h. The identifier can be read out via the object cob_id_sync.

Index	**1005_h**			
Name	**cob_id_sync**			
Info	--	rw	~~PDO~~	UINT32
Value	80_h	80_h		

6.6 Network Management (NMT service)

All CANopen devices can be controlled via the network management. The identifier with the highest priority (000h) is reserved for this purpose. Commands can be sent to one or all servo drives via NMT. Each command consists of two bytes, whereby the first byte contains the command code (command specifier, CS) and the second byte the node address (node id, NI) of the addressed servo drive. If zero is specified as node address, all nodes in the network will be addressed (broadcast). This makes it possible, for example, to trigger a reset in all devices at the same time. The servo drives do not acknowledge the NMT commands. It can only be concluded indirectly (e.g. by the Bootup message after a reset) that the reset was carried out successfully.

Structure of the NMT message:



States are defined in a state diagram for the NMT status of the CANopen node. State changes can be triggered via the CS byte in the NMT message. These are essentially oriented on the target state.

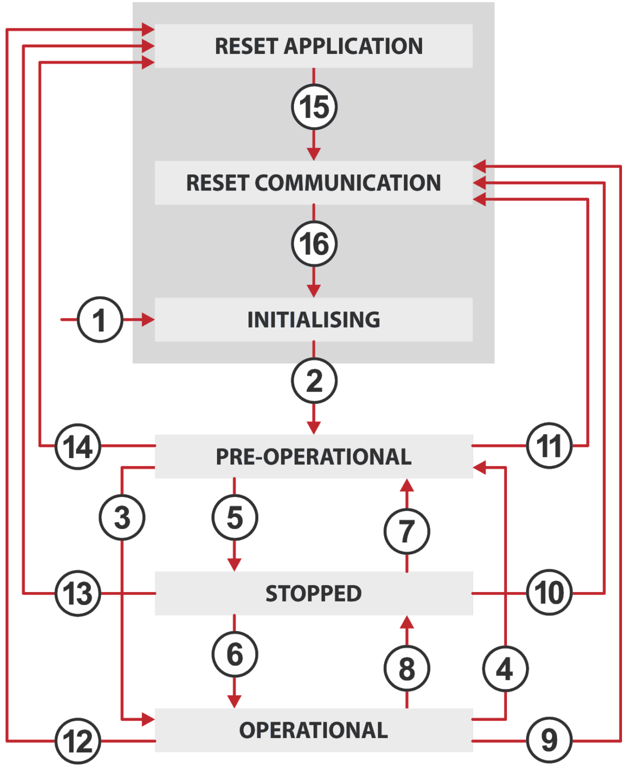


Figure 29: NMT-State machine

Transition	Name	CS	Target state	NMT state
1	Power on			
2	Bootup		Pre-Operational	$7F_h$
3	Start Remote Node	01_h	Operational	05_h
4	Enter Pre-Operational	80_h	Pre-Operational	$7F_h$
5	Stop Remote Node	02_h	Stopped	04_h
6	Start Remote Node	01_h	Operational	05_h
7	Enter Pre-Operational	80_h	Pre-Operational	$7F_h$
8	Stop Remote Node	02_h	Stopped	04_h
9	Reset Communication	82_h	Pre-Operational	$7F_h$
10	Reset Communication	82_h	Pre-Operational	$7F_h$
11	Reset Communication	82_h	Pre-Operational	$7F_h$
12	Reset Application	81_h	Pre-Operational	$7F_h$
13	Reset Application	81_h	Pre-Operational	$7F_h$
14	Reset Application	81_h	Pre-Operational	$7F_h$

State transitions 2, 15 and 16 are executed automatically by the servo drive when initialization is complete.

Depending on the NMT status, certain communication objects cannot be used: For example, it is absolutely necessary to set the NMT status to Operational so that the servo drive sends PDOs.

State	Description	SDO	PDO	NMT
Reset Application	No communication. All CAN objects are reset to their reset values (application parameter set).	-	-	-
Reset Communication	No communication. The CAN controller is reinitialised.	-	-	-
Initialising	State after hardware reset. Resetting the CAN node, sending the bootup message.	-	-	-
Pre-Operational	Communication via SDOs possible. PDOs not active (No sending / evaluation).	X	-	X
Operational	Communication via SDOs possible. All PDOs active (sending / evaluating).	X	X	X
Stopped	No communication except heartbeating.	-	-	X

INFORMATION **Note the following instructions**

- NMT telegrams must not be sent in a burst (one immediately after the other).
- There must be at least twice the position controller cycle time between two successive NMT telegrams on the bus (even for different nodes!) so that the servo drive can process the NMT telegrams correctly.
- The NMT command "Reset Application" is delayed, if necessary, until a running save operation is completed, as otherwise the save operation would remain incomplete (Defective parameter set). The delay can be in the range of a few seconds.
- The communication status must be set to Operational for the servocontroller to send and receive PDOs.

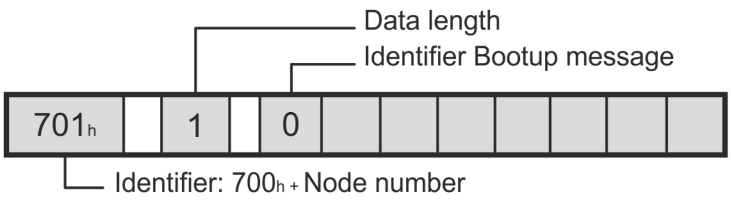
6.7 Bootup

6.7.1 Overview

After switching on the power supply or after a reset, the servo drive reports via a bootup message that the initialization phase has been completed. The servo drive then has the NMT status Pre-Operational.

6.7.2 Structure of the Bootup message

The bootup message is structured almost identically to the following heartbeat message. Only a zero is sent instead of the NMT status.



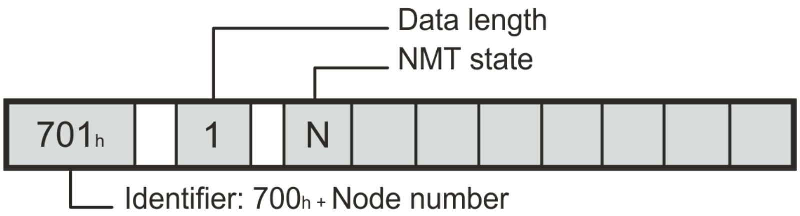
6.8 Heartbeat (Error Control Protocol)

6.8.1 Overview

To monitor the communication between slave (drive) and master, the so-called heartbeat protocol can be activated: The drive sends cyclic messages to the master. The master can check the cyclic occurrence of these messages and initiate appropriate measures if they fail to appear. Since both heartbeat and nodeguarding telegrams (see section 6.9 *Nodeguarding (Error Control Protocol)* on page 182) are sent with the identifier **700_h + node number**, both protocols cannot be active at the same time. If both protocols are activated at the same time, only the heartbeat protocol is active.

6.8.2 Structure of the Heartbeat message

The heartbeat telegram is sent with the identifier **700_h + node number**. It contains only 1 byte of user data, the NMT status of the servo drive (see section 6.6 *Network Management (NMT service)* on page 177).



NMT-State	State name
04_h	Stopped
05_h	Operational
$7F_h$	Pre-Operational

6.8.3 Description of objects

Object 1017_h: producer_heartbeat_time

To activate the heartbeat functionality, the time between two heartbeat telegrams can be defined via the object producer_heartbeat_time.

Index	**1017_h**			
Name	**producer_heartbeat_time**			
Info	ms	rw	~~PDO~~	UINT16
Value	0...65536	0		

The producer_heartbeat_time can be stored in the parameter set. If the servo drive starts with a producer_heartbeat_time not equal to zero, the bootup message is considered the first heartbeat. The servo drive can only be used as a heartbeat producer. Object 1016_h (consumer_heartbeat_time) is therefore only implemented for compatibility reasons and always returns 0.

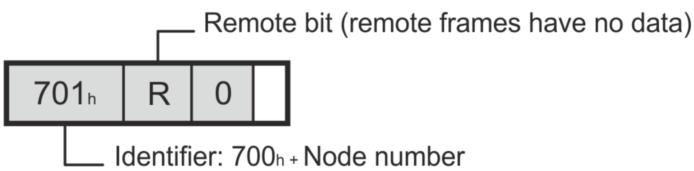
6.9 Nodeguarding (Error Control Protocol)

6.9.1 Overview

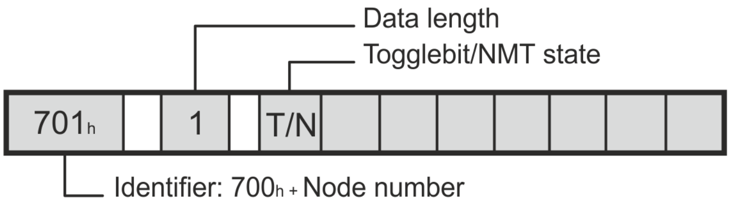
The nodeguarding protocol can also be used to monitor the communication between slave (drive) and master. In contrast to the heartbeat protocol, with nodeguarding the master and slave monitor each other: The master cyclically asks the drive for its NMT status. A certain bit is toggled in each response from the servo drive. If these answers are missing or if the servo drive always answers with the same toggle bit, the master can react accordingly. Similarly, the drive monitors the regular arrival of nodeguarding requests from the master: if the messages remain missing for a certain period of time, the servo drive triggers error 12-4. Since both heartbeat and nodeguarding telegrams (see section 6.9 *Nodeguarding (Error Control Protocol)* on page 182) are sent with the identifier **700_h + node number**, both protocols cannot be active at the same time. If both protocols are activated at the same time, only the heartbeat protocol is active.

6.9.2 Structure of the Nodeguarding messages

The request of the master must be sent as remote frame with the identifier **700h + node number**. With a remote frame a special bit is additionally set in the telegram, the remote bit. Remote frames have in principle no data.



The servo drive response is structured in the same way as the heartbeat message. It contains only 1 byte of user data, the togglebit and the NMT status of the servo drive.



Bit	Value	Name	Description
7	80_h	toggle_bit	Changes with every telegram
0...6	$7F_h$	nmt_state	section 6.6 *Network Management (NMT service)* on page 177

The monitoring time for requests from the master can be parameterised. The monitoring starts with the first received remote request of the master. From this point on, the remote requests must arrive before the set monitoring time has elapsed, otherwise error 12-4 is triggered. The togglebit is reset by the NMT command Reset Communication. It is therefore deleted in the first response of the servo drive.

6.9.3 Description of objects

6.9.3.1 Object $100C_h$: guard_time

To activate the node guarding monitoring, the maximum time between two remote queries of the master is parameterised. This time is determined in the servo drive from the product of guard_time ($100C_h$) and life_time_factor ($100D_h$). It is therefore recommended to describe the life_time_factor with 1 and then to specify the time directly via the guard_time in milliseconds.

Index	**$100C_h$**			
Name	**guard_time**			
Info	ms	rw	~~PDO~~	UINT16
Value	0...65536	0		

6.9.3.2 Object $100D_h$: life_time_factor

The life_time_factor should be set to 1 to specify the guard_time directly.

Index	**$100D_h$**			
Name	**life_time_factor**			
Info	--	rw	~~PDO~~	UINT8
Value	0...1	0		

6.10 Table of identifiers

The following table gives an overview of the identifiers used:

Object type	Identifier (hexadecimal)	Remark
SDO (Host to Servo)	600_h+ Node number	
SDO (Servo to Host)	580_h + Node number	
TPDO1	181_h / 180_h + Node number	These are the default values. The node number can be added automatically if the corresponding option is set (see section 2.1.5 *Activate CANopen* on page 16).
TPDO2	281_h / 280_h + Node number	
TPDO3	381_h / 380_h + Node number	
TPDO4	481_h / 480_h + Node number	
RPDO1	201_h / 200_h + Node number	
RPDO2	301_h / 300_h + Node number	
RPDO3	401_h / 400_h + Node number	
RPDO4	501_h / 500_h + Node number	
SYNC	080_h	
EMCY	080_h +Node number	
HEARTBEAT	700_h +Node number	
NODEGUARDING	700_h +Node number	
BOOTUP	700_h +Node number	
NMT	000_h	

7 Appendix

7.1 CANopen

CANopen is a standard developed by the association "CAN in Automation". A large number of device manufacturers are organised in this association. This standard has now replaced all manufacturer-specific CAN protocols.

The relevant elements of the object dictionary and the associated access methods are described in this CANopen manual.

CAN in Automation (CiA)
Kontumazgarten 3
DE-90429 Nürnberg
Tel.: +49-911-928819-0
Fax: +49-911-928819-79
headquarters(at)can-cia.org
www.can-cia.de

The CANopen implementation of the servo drive is based on the following standards:

- CiA Draft Standard 301, Version 4.02, 13. Februar 2002
- CiA Draft Standard Proposal 402, Version 2.0, 26. Juli 2002

7.2 Characteristics of the CAN interface

The CAN interface has the following characteristics:

- CAN specification V2.0 Part A (Part B passive, i.e. messages of this type are tolerated but not processed)
- Physical layer: ISO 11898

7.3 Error codes of the EMERGENCY message

CAN code	Error number	Description
2300_h	31-x	Group 31: I²t
2311_h	31-1	I²t servo drive
2312_h	31-0	I²t motor
2313_h	31-2	I²t PFC
2314_h	31-3	I²t braking resistor
2320_h	6-x	Group 6: Short circuit in the power output stage
3200_h	32-x	Group 32: PFC
3210_h	7-x	Group 7: Overvoltage
3220_h	2-x	Group 2: Undervoltage
3280_h	32-0	DC bus circuit charging time exceeded
3281_h	32-1	Undervoltage for active PFC
3282_h	32-5	Brake chopper overload. DC bus circuit could not be discharged.
3283_h	32-6	Discharging period DC bus circuit exceeded
3284_h	32-7	Supply voltage missing for enabling
3285_h	32-8	Supply voltage breakdown while servo drive enabled
3286_h	32-9	Phase failure
4200_h	4-x	Group 4: Overtemperature
4210_h	4-0	Overtemperature power output stage
4280_h	4-1	Overtemperature DC bus circuit
4310_h	3-x	Group 3: Overtemperature motor
5080_h	90-x	Group 90: HW initialisation
5110_h	5-x	Group 5: Internal voltage supply
5114_h	5-0	Failure of internal voltage 1
5115_h	5-1	Failure of internal voltage 2
5116_h	5-2	Driver supply failure
5200_h	21-x	Group 21: Current measurement
5220_h	16-4	Unexpected hardware error
5280_h	21-0	Error 1 current measurement U
5281_h	21-1	Error 1 current measurement V
5282_h	21-2	Error 2 current measurement U
5283_h	21-3	Error 2 current measurement V
5410_h	5-3	Undervoltage digital I/Os
5410_h	5-4	Overcurrent digital I/Os
5430_h	24-x	Group 24: Analogue input monitoring

CAN code	Error number	Description
5500_h	26-x	Group 26: Flash
5580_h	26-0	No user parameter set
5581_h	26-1	Checksum error
5582_h	26-2	Flash: write error
5583_h	26-3	Flash: erase error
5584_h	26-4	Flash: error in internal flash
5585_h	26-5	No calibration data
5586_h	26-6	No user position data sets
6000_h	25-x	Group 25: Invalid device type
6000_h	91-x	Group 91: SW initialisation
6080_h	25-0	Invalid device type
6081_h	25-1	Device type not supported
6082_h	25-2	Hardware revision not supported
6083_h	25-3	Device functionality restricted
6100_h	16-x	Group 16: Program execution
6180_h	1-x	Group 1: Stack overflow
6181_h	16-0	Incorrect program execution
6182_h	16-1	Illegal interrupt
6183_h	16-3	Unexpected state
6184_h	15-x	Group 15: Mathematics
6185_h	15-0	Division by zero
6186_h	15-1	Out of range error
6187_h	16-2	Initialisation error
6188_h	82-x	Group 82: Internal sequence control
6320_h	36-x	Group 36: Parameters
6380_h	30-x	Group 30: Internal calculations
7122_h	14-x	Group 14: Motor and resolver identification
7300_h	8-x	Group 8: Angle encoder
7380_h	8-0	Resolver/Hall angle encoder error
7382_h	8-2	Incremental encoder: Z0 track signals error
7383_h	8-3	Incremental encoder: Z1 track signals error
7384_h	8-4	Digital incremental encoder: track signals error
7385_h	8-5	Incremental encoder: hall signals error
7386_h	8-6	Angle encoder communication error
7387_h	8-7	Master frequency input: Incorrect signal amplitude incremental track
7388_h	8-8	Internal angle encoder error

CAN code	Error number	Description
7389_h	8-9	Encoder at [X2B/X6] not supported
$73A0_h$	9-x	Group 9: Encoder parameter set
$73A1_h$	9-0	Encoder parameter set: out-of-date format
$73A2_h$	9-1	Encoder parameter set cannot be decoded
$73A3_h$	9-2	Encoder parameter set: unknown version
$73A4_h$	9-3	Encoder parameter set: corrupted data structure
$73A5_h$	9-7	Encoder EEPROM is write protected
$73A6_h$	9-9	Too small memory size of encoder EEPROM
7580_h	60-x	Group 60: Ethernet
7581_h	61-x	Group 61: Ethernet
8000_h	45-x	Group 45: IGBT driver supply
8080_h	43-x	Group 43: HW limit switches
8081_h	43-0	Limit switch: negative setpoint inhibited
8082_h	43-1	Limit switch: positive setpoint inhibited
8083_h	43-2	Limit switch: positioning suppressed
8084_h	45-0	Driver supply cannot be switched off
8085_h	45-1	Driver supply cannot be switched on
8086_h	45-2	Driver supply has been activated
8090_h	51-x	Group 51: FSM 2.0
8091_h	51-0	No / unknown FSM module or driver supply faulty
8093_h	51-2	FSM: unequal module type
8094_h	51-3	FSM: unequal module version
8095_h	51-4	FSM: error in SSIO communication
8096_h	51-5	FSM: error in brake activation
8097_h	51-6	FSM: unequal serial number
8098_h	52-x	Group 52: FSM 2.0 STO
8099_h	52-1	FSM: discrepancy time expired
$809A_h$	52-2	FSM: STOA/STOB deactivated while power output stage enabled
$809B_h$	52-3	FSM: Limitation error
$80A0_h$	53-x	Group 53: FSM: Violation of safety conditions
$80A1_h$	53-0	USF0: safety condition violated
$80A2_h$	53-1	USF1: safety condition violated
$80A3_h$	53-2	USF2: safety condition violated
$80A4_h$	53-3	USF3: safety condition violated
$80A9_h$	54-x	Group 54: FSM: Violation of safety conditions
$80AA_h$	54-0	SBC: safety condition violated

CAN code	Error number	Description
$80AC_h$	54-2	SS2: safety condition violated
$80AD_h$	54-3	SOS: safety condition violated
$80AE_h$	54-4	SS1: safety condition violated
$80AF_h$	54-5	STO: safety condition violated
$80B0_h$	54-6	SBC: brake not released for > 10 days
$80B1_h$	54-7	SOS: SOS requested for > 10 days
$80C0_h$	55-x	Group 55: FSM: Actual value evaluation 1
$80C1_h$	55-0	FSM: no actual speed / position value available or standstill for > 10 days
$80C2_h$	55-1	FSM: SINCOS encoder [X2B] - signal error
$80C3_h$	55-2	FSM: SINCOS encoder [X2B] - standstill > 10 days
$80C4_h$	55-3	FSM: Resolver [X2A] - signal error
$80C6_h$	55-7	FSM: other encoder [X2B] - Faulty angle information
$80C7_h$	55-8	FSM: impermissible acceleration detected
$80D0_h$	56-x	Group 56: FSM: Actual value evaluation 2
$80D1_h$	56-8	FSM: speed / angle difference encoder 1 - 2
$80D2_h$	56-9	FSM: error cross comparison encoder evaluation
$80E0_h$	57-x	Group 57: FSM: Inputs/Outputs
$80E1_h$	57-0	FSM: I/O - Self test error (internal/external)
$80E2_h$	57-1	FSM: digital inputs - signal level error
$80E3_h$	57-2	FSM: digital inputs - test pulse error
$80E7_h$	57-6	FSM: overtemperature
$80E8_h$	58-x	Group 58: FSM: Communication / Parameterisation
$80E9_h$	58-0	FSM: plausibility check of parameters
$80EA_h$	58-1	FSM: general error parameterisation
$80ED_h$	58-4	FSM: buffer internal communication
$80EE_h$	58-5	FSM: communication safety module - servo drive
$80EF_h$	58-6	FSM: error in cross comparison for processors 1 - 2
$80F0_h$	59-x	Group 59: FSM: Internal Error
$80F1_h$	59-1	FSM: failsafe supply / safe pulse inhibitor
$80F2_h$	59-2	FSM: error external power supply
$80F3_h$	59-3	FSM: error internal power supply
$80F4_h$	59-4	FSM: error management: too many errors
$80F5_h$	59-5	FSM: error writing to permanent event memory
$80F6_h$	59-6	FSM: error on saving parameter set
$80F7_h$	59-7	FSM: flash checksum error
$80F8_h$	59-8	FSM: internal monitoring, processor 1 - 2

CAN code	Error number	Description
$80F9_h$	59-9	FSM: other unexpected error
8100_h	12-x	Group 12: CAN communication
8100_h	13-x	Group 13: Timeout CAN bus
8120_h	12-1	CAN: communication error, bus OFF
8130_h	12-4	CAN: Node Guarding
8180_h	12-0	CAN: duplicate node number
8181_h	12-2	CAN: communication error (sending)
8182_h	12-3	CAN: communication error (receiving)
8183_h	12-9	CAN: protocol error
8184_h	13-0	Timeout CAN bus
8200_h	50-x	Group 50: CAN communication
8210_h	12-5	CAN: RPDO too short
8480_h	35-x	Group 35: Linear motor
8600_h	42-x	Group 42: Positioning
8611_h	17-x	Group 17: Max. following error exceeded
8611_h	27-x	Group 27: Following error monitoring
8612_h	40-x	Group 40: SW limit switches
8680_h	42-0	Positioning: no follow-up position: stop
8681_h	42-1	Positioning: reversal of rotation not permissible: stop
8682_h	42-2	Positioning: reversal of rotation after stop not permissible
8700_h	34-x	Group 34: Fieldbus
8780_h	34-0	No synchronisation via fieldbus
8781_h	34-1	Fieldbus synchronisation error
$8A00_h$	11-x	Group 11: Homing run
$8A00_h$	33-x	Group 33: Following error encoder emulation
$8A80_h$	11-0	Error when homing run is started
$8A81_h$	11-1	Error during homing run
$8A82_h$	11-2	Homing: no valid index pulse
$8A83_h$	11-3	Homing: timeout
$8A84_h$	11-4	Homing: incorrect / invalid limit switch
$8A85_h$	11-5	Homing: I²t / following error
$8A86_h$	11-6	Homing: end of search distance reached
$8A87_h$	33-0	Following error encoder emulation
$F000_h$	80-x	Group 80: IRQ_0_3
$F080_h$	80-0	Time overflow current control IRQ
$F081_h$	80-1	Time overflow speed control IRQ

CAN code	Error number	Description
$F082_h$	80-2	Time overflow position control IRQ
$F083_h$	80-3	Time overflow interpolator IRQ
$F084_h$	81-4	Time overflow low-level IRQ
$F085_h$	81-5	Time overflow MDC IRQ
$FF00_h$	28-x	Group 28: Operating hours meter
$FF01_h$	28-0	Missing operating hours meter
$FF02_h$	28-1	Operating hours meter: write error
$FF03_h$	28-2	Operating hours meter corrected
$FF04_h$	28-3	Operating hours meter converted