# FAULHABER

Communications
Manual

| | |
|---|---|
| MC 5010 | MCS |
| MC 5005 | MC 3001 |
| MC 5004 | MC 3603 |
| MC 5004 P STO | IMC |

# CANopen®

Version:
7th edition, 2.02.2024

# Content

# Content

# 1 About this document

## 1.1 Validity of this document

This document describes:

- Communication with the drive via CANopen
- Basic services provided by the Communication structure
- Methods for accessing the parameters
- Drive from the viewpoint of the communication system

This document is intended for software developers with CAN-BUS experience, and for CAN-BUS project engineers.

All data in this document relate to the standard versions of the drives. Changes relating to customer-specific versions can be found in the corresponding data sheet.

All data in this document relate to the firmware revision M.

## 1.2 Associated documents

For certain actions during commissioning and operation of FAULHABER products additional information from the following manuals is useful:

| Manual | Description |
|---|---|
| Motion Manager 6 | Operating instructions for FAULHABER Motion Manager PC software |
| Quick start guide | Description of the first steps for commissioning and operation of FAULHABER Motion Controllers |
| Drive functions | Description of the operating modes and functions of the drive |
| Technical manual | Instructions for installation and use of the FAULHABER Motion Controller |
| CiA 301 | CANopen application layer and communication profile |
| CiA 402 | CANopen device profile for drives and motion control |

These manuals can be downloaded in pdf format from the web page www.faulhaber.com/manuals.

## 1.3 Using this document

- Read the document carefully before undertaking configuration.
- Retain the document throughout the entire working life of the product.
- Keep the document accessible to the operating personnel at all times.
- Pass the document on to any subsequent owner or user of the product.

## 1.4 List of abbreviations

| Abbreviation | Meaning |
|---|---|
| Attr. | Attribute |
| CAN | Controller Area Network |
| CiA | CAN in Automation e.V. |
| COB ID | Communication Object Identifier |
| CS | Command Specifier |
| EEPROM | Electrically Erasable Programmable Read-Only Memory |
| EMCY | Emergency |
| HB | High Byte |
| HHB | Higher High Byte |
| HLB | Higher Low Byte |
| LB | Low Byte |
| LHB | Lower High Byte |
| LLB | Lower Low Byte |
| LSB | Least Significant Byte |
| LSS | Layer Setting Service |
| MSB | Most Significant Byte |
| NMT | CANopen network management |
| OD | Object dictionary |
| PDO | Process Data Object |
| PP | Profile Position |
| PV | Profile Velocity |
| ro | read only |
| RTR | Remote Request |
| rw | read-write |
| RxPDO | Receive Process Data Object (PDO received from the drive) |
| SDO | Service Data Object |
| PLC | Programmable Logic Controller |
| Sxx | Data type signed (negative and positive numbers) with bit size xx |
| SYNC | Synchronization object |
| TxPDO | Transmit Process Data Object (PDO sent from the drive) |
| Uxx | Data type unsigned (positive numbers) with bit size xx |

FAULHABER

## 1.5    Symbols and designations

⚠ **NOTICE!**
**Risk of damage.**
   ▸    Measures for avoidance

ℹ    Instructions for understanding or optimizing the operational procedures

   ✓    Pre-requirement for a requested action
   1.    First step for a requested action
      ↳    Result of a step
   2.    Second step of a requested action
   ↳    Result of an action
   ▸    Request for a single-step action

# 2    Overview

## 2.1    Basic structure of a CANopen device
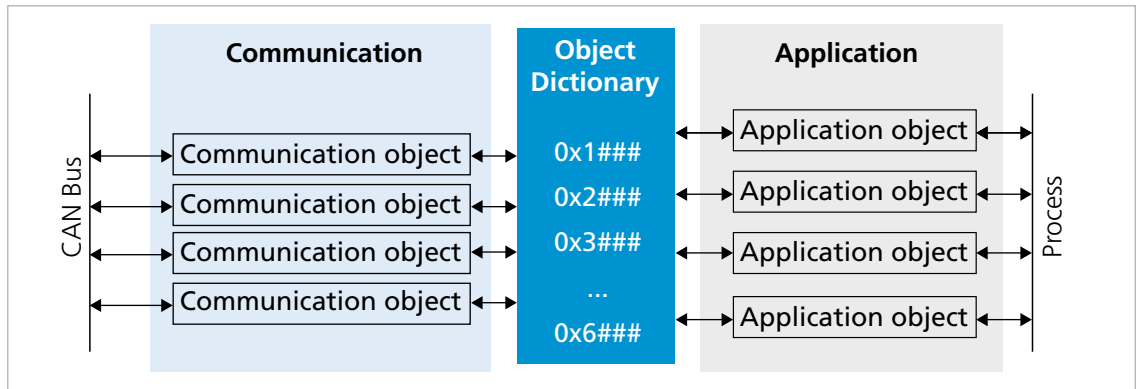
Fig. 1:    Basic structure of a CANopen device

**Communication services**

The CANopen master communicates with the object dictionary via the bus system and using the communication services (see chap. 3.2, p. 15).

**Object dictionary**

The object dictionary contains parameters, set-points and actual values of a drive. The object dictionary is the link between the application (drive functions) and the communication services. All objects in the object dictionary can be addressed by a 16-bit index number (0x1000 to 0x6FFF) and an 8-bit subindex (0x00 to 0xFF).

| Index | Assignment of the objects |
|---|---|
| 0x1000 to 0x1FFF | Communication objects |
| 0x2000 to 0x5FFF | Manufacturer-specific objects |
| 0x6000 to 0x6FFF | Objects of the drive profile acc. to CiA 402 |

The values of the parameters can be changed by the communication side or by the drive side.

**Application part**

The application part contains drive functions corresponding to CiA 402. The drive functions read parameters from the object dictionary, obtain the set-points from the object dictionary and return actual values. The parameters from the object dictionary determine the behavior of the drive.

> No further details of the application part are given in this document. The communication with the drive and the associated operating modes are described in the separate "Drives Functions" manual.

## 2.2 Requirements for communication

FAULHABER drives are delivered in the unconfigured state. For operation in a CAN network, a unique node number must be assigned and a baud rate must be set during initial commissioning (see chap. 4, p. 35).

After switching on and initializing, the Motion Controller is at first in the **Pre-Operational** state. In order to be able to perform drive functions, the Motion Controller must be brought into the **Operational** state (see chap. 3.8, p. 27).

1. Connect the controller to a voltage supply (supply at least for the electronics).

2. Connect CAN_H, CAN_L, GND to the respective terminals of a host-side CAN connection.

3. Switch on the voltage and establish a connection via the configuration application.
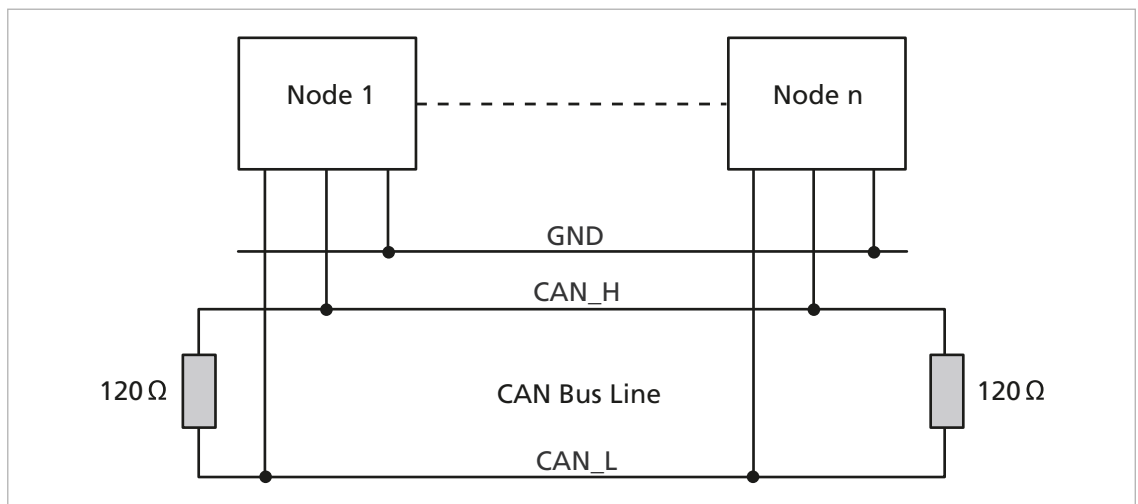


Fig. 2: Connection to the CANopen network

## 2.3    FAULHABER Motion Manager

We recommend that the first commissioning of a FAULHABER drive is performed using the "FAULHABER Motion Manager" software.

The FAULHABER Motion Manager enables simple access to the settings and parameters of the connected motor controllers. The graphical user interface allows configurations to be read, changed and reloaded. Individual commands or complete parameter sets and program sequences can be input and loaded to the controller.

Wizard functions support the user when commissioning the drive controllers. The wizard functions are arranged on the user interface in the sequence they are normally used:

- Connection wizard: Supports the user in setting up the connection to the connected controller
- Motor wizard: Supports the user in adapting an external controller to the connected motor by selecting the respective FAULHABER motor
- Controller setting wizard: Supports the user in optimizing the controller parameters.

The software can be downloaded free of charge from the FAULHABER website: https://www.faulhaber.com/motionmanager.

**i**    We recommend always using the latest version of the FAULHABER Motion Manager.

The FAULHABER Motion Manager is described in the separate "Motion Manager 6" manual. The contents of the manual are also available as context-sensitive online help within the FAULHABER Motion Manager.

FAULHABER

## 2.4 Saving and restoring parameters

So that changed parameters in the OD remain active in the controller when it is switched on again, the "Save" command must be executed to save them permanently in the non-volatile memory (EEPROM application) (see chap. 5.1, p. 38). When the motor is switched on, the parameters are loaded automatically from the non-volatile memory into RAM.
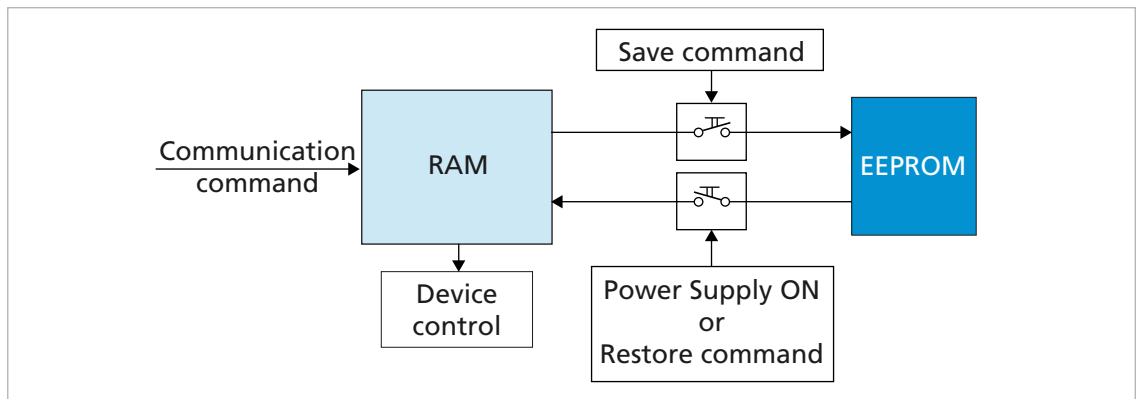


Fig. 3: Saving and restoring parameters

The following parameters can be loaded using the "Restore" command (see chap. 5.1, p. 38):

- Factory settings
- Parameters saved using the "Save" command

### 2.4.1 Save parameters

The current parameter settings can be saved in the internal EEPROM (SAVE) (see Tab. 17), either completely or for individual ranges.

▶ Write the "save" signature to the subindex 01 to 05 of the object 0x1010 (see Tab. 18).

### 2.4.2 Restoring settings

ℹ When the drive is switched on, the saved parameters are loaded automatically.

Factory settings or last saved parameter settings can be loaded from the internal EEPROM at any time, completely or for specific ranges (RESTORE) (see Tab. 19).

1. Write the "Load" signature to the subindex 01 to 06 of the object 0x1011 (see Tab. 20).

   ⮑ After Restore Factory (01), Restore Communication (02) and Restore Application (03), the drive must be reset. Only then are the parameters updated.

2. Application parameters (04), together with record 1 and record 2 of the special application parameters (05/06) can be updated with the "Reload" command.

   ⮑ The "Reload" command overwrites the values last saved as application parameters.

ℹ If it is desired that the values currently loaded remain available after a "Restore", these must be saved to the PC using a suitable program (such as FAULHABER Motion Manager).

### 2.4.3    Changing the parameter set

The repository for the application parameters (motor data, I/O configuration, controller parameters, operating mode, etc.) includes a common basic set of parameters (App) as well as a storage area for parameters which often need to be adapted to variations in the load situation (App1/App2):

**Speed controller and filter**

| Index | Subindex | Name | Type | Attr. | Meaning |
|---|---|---|---|---|---|
| 0x2344 | 0x01 | Gain $K_P$ | U32 | rw | Controller gain [As $1e^{-6}$] |
| | 0x02 | Integral time TN | U16 | rw | Controller reset time [100 µs] |
| 0x2346 | 0x01 | Set Point Velocity Filter Time T_F | U16 | rw | Filter time T_F [100 µs] |
| | 0x02 | Setpoint Filter Enable | U8 | rw | 0: inactive<br>1: Active |
| 0x2347 | 0x01 | Gain Factor | U8 | rw | Gain factor (used by the speed control in PP mode on the $K_P$)<br>0: The gain factor of the speed controller is reduced to 0 at the target<br>128: no variable gain<br>255: The gain factor of the speed controller is doubled at the target |

**Position controller**

| Index | Subindex | Name | Type | Attr. | Meaning |
|---|---|---|---|---|---|
| 0x2348 | 0x00 | Number of entries | U8 | ro | Number of object entries |
| | 0x01 | $K_v$ [1/s] | U8 | rw | Range: 1-250 |

**Pre-controls**

| Index | Subindex | Name | Type | Attr. | Meaning |
|---|---|---|---|---|---|
| 0x2349 | 0x01 | Torque/force feed forward factor | U8 | rw | Factor for the torque or force control<br>0: 0% activation of the feedforward value<br>128: 100% feedforward control |
| | 0x02 | Torque/Force feed forward delay | U8 | rw | Set-point delay:<br>0: undelayed activation<br>1: Activation delayed by one sampling |
| 0x234A | 0x01 | Velocity feed forward factor | U8 | rw | Factor for the torque or force control<br>0: 0% feedforward control<br>128: 100% feedforward control |
| | 0x02 | Velocity feed forward delay | U8 | rw | Set-point delay:<br>0: undelayed activation<br>1: Activation delayed by one sampling |

# Overview

**General settings**

| Index | Subindex | Name | Type | Attr. | Meaning |
|---|---|---|---|---|---|
| 0x6060 | 0x00 | Modes of Operation | S8 | rw | Select the operating mode<br>−4: ATC<br>−3: AVC<br>−2: APC<br>−1: Voltage mode<br>0: Controller not activated<br>1: PP<br>3: PV<br>6: Homing<br>8: CSP<br>9: CSV<br>10: CST |
| 0x6081 | 0x00 | Profile Velocity | U32 | rw | Profile velocity in user-defined units |
| 0x6083 | 0x00 | Profile acceleration | U32 | rw | Profile acceleration $[1/s^2]$ |
| 0x6084 | 0x00 | Profile deceleration | U32 | rw | Profile deceleration $[1/s^2]$ |
| 0x6086 | 0x00 | Motion Profile Type | S16 | rw | Speed profile type:<br>0: Linear profile<br>1: $Sin^2$ speed |
| | | | | | |
| 0x60E0 | 0x00 | Positive torque limit value | U16 | rw | Value of the upper limit value [in relative scaling] |
| 0x60E1 | 0x00 | Negative torque limit value | U16 | rw | Value of the lower limit value [in relative scaling] |

These parameters are stored twice. During operation, the system can switch quickly between these different presets.

**Create an application set**

▶ Save application parameters 1: Write the "save" signature to subindex 04 of object 0x1010.

    ↪ The current data is saved as the application parameter set 1.

▶ Save application parameters 2: Write the "save" signature to subindex 05 of object 0x1010.

    ↪ The current data is saved as the application parameter set 2.

**Activate an application set**

▶ Reload application parameters 1: Write the "load" signature to subindex 05 of object 0x1011.

    ↪ Current data from the application parameter set 1 is activated directly.

▶ Reload application parameters 2: Write the "load" signature to subindex 06 of object 0x1011.

    ↪ Current data from the application parameter set 2 is activated directly.

FAULHABER

# 3 CANopen protocol description

## 3.1 Introduction

**CANopen**

CANopen is a standard software protocol. A CAN hardware environment is required for communication using CANopen. Up to 127 nodes can be addressed within a CANopen network. The maximum transmission speed is 1 MBit/s.

**CAN standardization**

The CiA defines the following aspects in CiA 301:

- Communication structure
- Control and monitoring functions

CANopen device profiles have been defined for a wide range of device classes, such as:

- CiA 402 for drives
- CiA 401 for input and output devices

**Structure of a CANopen telegram**

A CANopen telegram has an 11-bit identifier and can contain up to 8 bytes of user data.

Tab. 1:    Schematic structure of a CANopen telegram

| 11-bit identifier | up to 8 bytes user data | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 11-bit | 8-bit | 8-bit | 8-bit | 8-bit | 8-bit | 8-bit | 8-bit | 8-bit |

## 3.2 Communication services
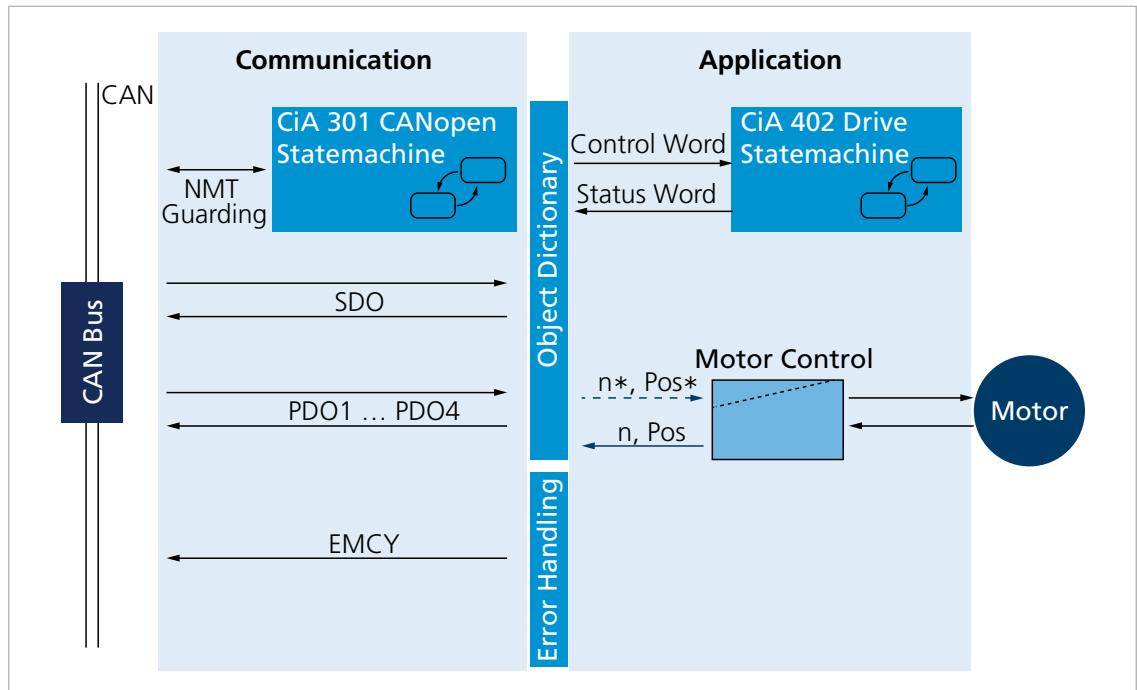


Fig. 4:    Communication services of the Motion Controller

The communication part contains communication services as specified in CiA 301.

Tab. 2:    Communication services to CiA 301

| Communication services | Description |
|---|---|
| NMT (Network Management) | Activates nodes and monitors the current status of a node (see chap. 3.8, p. 27). |
| SDO (Service Data Object) | The CANopen master uses the SDO to access parameters within a node. Each SDO access reads or writes exactly one parameter. An SDO can only address one node in a network (see chap. 3.5, p. 21). |
| PDO (Process Data Object) | The PDO is used to access real-time data. A PDO can use a CAN message to access multiple drive parameters concurrently. The parameters sent or received in a PDO can be freely configured (see chap. 3.4, p. 17). |
| SYNC object | SYNC objects are used to synchronize different applications on the CAN-BUS (see chap. 3.7, p. 26). |
| EMCY (Emergency Object) | An emergency message is used to inform the CANopen master about errors. A CAN message conveys the error code asynchronously so that the status of the CANopen slave need not be interrogated after an error (see chap. 3.6, p. 24). |

**Communication profile**
FAULHABER Motion Controllers support the CANopen communications profile to CiA 301 V4:

- 4 transmission PDOs
- 4 receipt PDOs
- 1 server SDO
- Emergency object
- NMT with node guarding and heartbeat
- SYNC object

FAULHABER

ℹ The data assignment of the PDOs is pre-set to the "PDO set for servo drive" as specified in CiA 402 V3, but can be changed by the user (dynamic PDO mapping).

## 3.3 Identifier distribution

The Communication Object Identifier (COB-ID) consists of a 7-bit node address (NodeID) and a 4-bit function code.
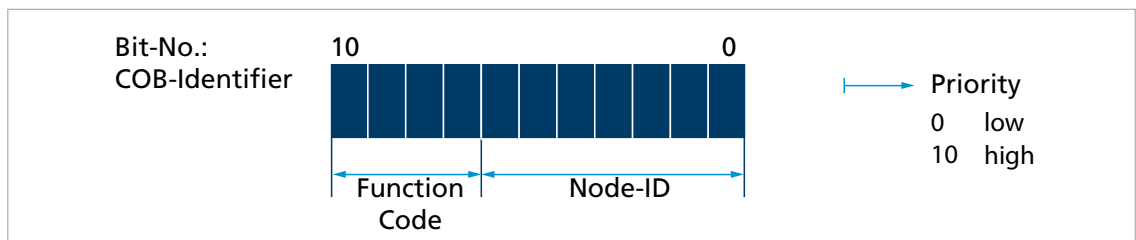


Fig. 5:     Identifier distribution

The Predefined-Connection-Set defines the standard identifier for the most important objects.

Tab. 3:     Standard identifier

| Object | Function code (binary) | Resulting COB-ID | Object index for communication setting |
|---|---|---|---|
| NMT | 0000 | 0 | – |
| SYNC | 0001 | 128 (80h) | 1005 h |
| EMERGENCY | 0001 | 129 (81h) to 255 (FFh) | 1014 h |
| PDO1 (tx) | 0011 | 385 (181h) to 511 (1FFh) | 1800 h |
| PDO1 (rx) | 0100 | 513 (201h) to 639 (27Fh) | 1400h |
| PDO2 (tx) | 0101 | 641 (281h) to 767 (2FFh) | 1801 h |
| PDO2 (rx) | 0110 | 769 (301h) to 895 (37Fh) | 1401h |
| PDO3 (tx) | 0111 | 897 (381h) to 1023 (3FFh) | 1802 h |
| PDO3 (rx) | 1000 | 1025 (401h) to 1151 (47Fh) | 1402h |
| PDO4 (tx) | 1001 | 1153 (481h) to 1279 (4FFh) | 1803 h |
| PDO4 (rx) | 1010 | 1281 (501h) to 1407 (57Fh) | 1403h |
| SDO (tx) | 1011 | 1409 (581h) to 1535 (5FFh) | 1200h |
| SDO (rx) | 1100 | 1537 (601h) to 1663 (67Fh) | 1200h |
| NMT error control | 1110 | 1793 (701h) to 1919 (77Fh) | – |

The COB-IDs of the PDOs, the SYNC objects and the emergency objects can be changed via the communication parameters in the object dictionary. The COB-ID of the SDO telegram cannot be changed and is always derived from the node number.

FAULHABER

As delivered the system is configured with the node number 1. The COB-IDs are pre-set correspondingly:

- RxPDO: 201h, 301h, 401h and 501h
- TxPDO: 181h, 281h, 381h and 481h
- EMCY: 81 h
- RxSDO: 581 h
- TxSDO: 601 h

If node number 255 (unconfigured CANopen node) is changed to a node number >127 via the LSS protocol, the COB-IDs that are dependent on the node number are adapted automatically (see chap. 4.1.3, p. 36).

## 3.4 PDO (Process Data Object)

PDOs are CAN messages with up to 8 bytes user data. PDOs contain process data for controlling and monitoring the behavior of the device. The drive makes the distinction between receipt PDOs and transmission PDOs.

- Receipt PDOs (RxPDO): are received by a drive and typically contain control data
- Transmission PDOs (TxPDO): are sent by a drive and typically contain monitoring data

PDOs are evaluated or transmitted only when the device is in the NMT *Operational* state (see chap. 3.8, p. 27).

The transmission of PDOs can be triggered in various different ways. The behavior can be set for each PDO via the transmission type parameter of the communication parameters in the object dictionary:

Tab. 4: Types of PDO transmissions

| Transmission Type | Description |
|---|---|
| Event-driven | Event-driven RxPDOs are processed immediately on receipt. |
| | Event-driven TxPDOs are sent when the statusword of the device is contained and has been changed. |
| Remote request (RTR) | Data are sent in response to a request message. |
| Synchronized | Data are sent after receipt of a SYNC object (see chap. 3.7, p. 26). |

### 3.4.1 PDO configuration

- A maximum of 4 parameters can be mapped in one PDO.

- The data assignment of PDOs can be changed via the objects 0x1600 to 0x1603 and 0x1A00 to 0x1A03. The mapping procedure necessary for this is described in CiA 301. A suitable tool (such as FAULHABER Motion Manager or the configuration tool for the PLC controller used) is necessary for the mapping procedure.

- The transmission type and COB-ID of the PDOs can be changed via the objects 0x1400 to 0x1403 and 0x1800 to 0x1803.

- The transmission type parameter can be used to set the behavior of a PDO:

Tab. 5: Transmission type of a PDO

| Transmission Type | Meaning |
|---|---|
| 0 | synchronous, acyclical<br><br>A PDO is sent or executed once after a SYNC object when the contents of the PDO have changed (see chap. 3.7, p. 26). |
| 1 to 240 | synchronous, cyclical<br><br>A PDO is sent after every SYNC object (see chap. 3.7, p. 26). The value is then equal to the number of SYNC objects that must be received before the PDO is sent again (1 = PDO is sent for every SYNC object) |
| 252 | Only with TxPDOs: asynchronous<br>▪ When a SYNC signal is received, the content of the TxPDO is saved<br>▪ When a request (RTR) is received, the TxPDO is sent to the master |
| 253 | Only with TxPDOs: asynchronous<br><br>When a request (RTR) is received, the TxPDO is sent to the master |
| 255 | asynchronous (event-driven) |

### 3.4.2 PDO mapping in the standard configuration (status as delivered)

**RxPDO1: Controlword**

| 11-bit identifier | 2 bytes user data | | |
|---|---|---|---|
| 0x200 (512d) + node ID | LB | HB | |

The RxPDO1 contains the 16-bit Controlword to CiA DSP402. The Controlword controls the state machine of the drive unit and points to the object index 0x6040 in the object dictionary. The bit distribution is described in the documentation for the drive functions.

**TxPDO1: Statusword**

| 11-bit identifier | 2 bytes user data | | |
|---|---|---|---|
| 0x180 (384d) + node ID | LB | HB | |

The TxPDO1 contains the 16-bit Statusword to CiA 402. The Statusword indicates the status of the drive unit an and points to the object index 0x6041 in the object dictionary. The bit distribution is described in the documentation for the drive functions.

**RxPDO2: Controlword, Target Position (PP)**

| 11-bit identifier | 6 bytes user data | | | | | |
|---|---|---|---|---|---|---|
| 0x300 (768d) + node ID | LB | HB | LLB | LHB | HLB | HHB |

The RxPDO2 contains the 16-bit Controlword and the 32-bit value of the target position (object 0x607A) for the Profile Position mode (PP).

**TxPDO2: Statusword, Position Actual Value**

| 11-bit identifier | 6 bytes user data | | | | | |
|---|---|---|---|---|---|---|
| 0x280 (640d) + node ID | LB | HB | LLB | LHB | HLB | HHB |

The TxPDO2 contains the 16-bit Statusword and the 32-bit value of the actual position (object 0x6064).

**RxPDO3: Controlword, Target Velocity (PV)**

| 11-bit identifier | 6 bytes user data | | | | | |
|---|---|---|---|---|---|---|
| 0x400 (1024d) + node ID | LB | HB | LLB | LHB | HLB | HHB |

The RxPDO3 contains the 16-bit controlword and the 32-bit value of the target speed (object 0x60FF) for the Profile Velocity mode (PV).

**TxPDO3: Statusword, Velocity Actual Value**

| 11-bit identifier | 6 bytes user data | | | | | |
|---|---|---|---|---|---|---|
| 0x380 (896d) + node ID | LB | HB | LLB | LHB | HLB | HHB |

The TxPDO3 contains the 16-bit statusword and the 32-bit value of the actual speed (object 0x606C).

**RxPDO4: Controlword, Target Torque**

| 11-bit identifier | 4 byte of user data | | | | | |
|---|---|---|---|---|---|---|
| 0x400 (1024d) + node ID | LB | HB | LLB | LHB | HLB | HHB |

The RxPDO4 contains the 16-bit controlword and the 16-bit value of the target torque (object 0x6071) for Cyclic Torque mode (CST).

**TxPDO4: Statusword, Torque Actual Value**

| 11-bit identifier | 4 byte of user data | | | | | |
|---|---|---|---|---|---|---|
| 0x380 (896d) + node ID | LB | HB | LLB | LHB | HLB | HHB |

The RxPDO4 contains the 16-bit statusword and the 16-bit value of the actual torque (object 0x6077) for Cyclic Torque mode (CST).

### 3.4.3 Dealing with mapping errors

If the mapping procedure specified in CiA 301 is not complied with, one of the following SDO errors will be returned:

Tab. 6: SDO errors in response the incorrect mapping procedure

| SDO error | Meaning | Cause |
|---|---|---|
| 0x06090030 | General value range error | The mapping parameter lies outside that specified in the mapping procedure. |
| 0x06020000 | Object not present in the object dictionary | The value for the number of mapped objects is greater than the number of valid entries in the respective subindexes for the mapping parameter objects. |

If the number of mapped objects is 0, the PDO will be flagged internally as invalid and will not be operated.

ℹ️ Other mapping errors are described in the SDO error table (see chap. 3.5.2, p. 23).

### 3.4.4 Dummy mappings

RxPDOs can be configured so that more than one participant can respond to them. In this case it may be desirable that only part of the data contained in the PDO is evaluated in one of the devices.

For data that is not used locally, a dummy mapping to one of the supported data types can be entered in the mapping table of the PDO:

| Index | Type |
|---|---|
| 0x0002 | S8 |
| 0x0003 | S16 |
| 0x0004 | S32 |
| 0x0005 | U8 |
| 0x0006 | U16 |
| 0x0007 | U32 |

**Example**
An RxPDO contains the target positions for two axes.

Mapping for the node that should respond to the first target position:

- 0x160x.00 = 2
- 0x160x.01 = 0x607A0020
- 0x160x.02 = 0x00040020

Mapping for the node that should respond to the second target position:

- 0x160x.00 = 2
- 0x160x.01 = 0x00040020
- 0x160x.02 = 0x607A0020

## 3.5 SDO (Service Data Object)

The SDO reads and writes parameters in the OD (object dictionary). The SDO accesses the object dictionary via the 16-bit index and the 8-bit subindex. At the request of the client (PC, PLC (programmable logic controller)) the Motion Controller makes data available (upload) or receives data from the client (download).

Tab. 7:     General structuring of the SDO user data

| Byte0 | Byte 1 to 2 | Byte 3 | Byte 4 to 7 |
|---|---|---|---|
| Command specifier | 16-bit index | 8-bit subindex | 4-byte parameter data |

Tab. 8:     Distribution of the SDO transfer types

| Transfer type | Number of bytes | Purpose |
|---|---|---|
| Expedited transfer | Maximum 4 bytes | Read and write individual numeric parameters |
| Segmented Transfer | More than 4 bytes | Read text parameters (such as device name, firmware version) and transmit data blocks (such as the trace buffer) |

Only the expedited transfer is described in this document. The segmented transfer is described in CiA 301.

### 3.5.1 Expedited transfer

SDO messages are always size 8 bytes.

**Read OD entries (Client-to-Server, Upload-Request)**

| 11-bit identifier | 8 bytes user data | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0x600 (1536d) + node ID | 0x40 | Index LB | Index HB | Subindex | 0 | 0 | 0 | 0 |

**Server-to-Client, Upload-Response**

| 11-bit identifier | 8 bytes user data | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0x580 (1408d) + node ID | CS(0x4x) | Index LB | Index HB | Subindex | LLB (D0) | LHB (D1) | HLB (D2) | HHB (D3) |

The command specifier CS(0x4x) specifies the number of valid data bytes in D0 to D3 and the transfer code. The command specifier is coded as follows:

- CS = 0x4F, 1 data byte in D0
- CS = 0x4B, 2 data bytes in D0 to D1
- CS = 0x47, 3 data bytes in D0 to D2
- CS = 0x43, 4 data bytes in D0 to D3

**Write OD entries (Client-to-Server, Download-Request)**

| 11-bit identifier | 8 bytes user data | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0x600 (1536d) + node ID | CS(0x2x) | Index LB | Index HB | Subindex | LLB (D0) | LHB (D1) | HLB (D2) | HHB (D3) |

The command specifier CS(0x2x) specifies the number of valid data bytes in D0 to D3 and the transfer code. The command specifier is coded as follows:

- CS = 0x2F, 1 data byte in D0

- CS = 0x2B, 2 data bytes in D0 to D1

- CS = 0x27, 3 data bytes in D0 to D2

- CS = 0x23, 4 data bytes in D0 to D3

- CS = 0x22, no specification of the number of data bytes

**Server-to-Client, Download-Response**

| 11-bit identifier | 8 bytes user data | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0x580 (1407d) + node ID | 0x60 | Index LB | Index HB | Subindex | 0 | 0 | 0 | 0 |

**Abort in the event of SDO errors**
SDO-abort Client-to-Server

| 11-bit identifier | 8 bytes user data | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0x600 (1536d) + node ID | 0x80 | Index LB | Index HB | Subindex | ERROR 0 | ERROR 1 | ERROR 2 | ERROR 3 |

SDO-abort Server-to-Client

| 11-bit identifier | 8 bytes user data | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0x580 (1536d) + node ID | 0x80 | Index LB | Index HB | Subindex | ERROR 0 | ERROR 1 | ERROR 2 | ERROR 3 |

### 3.5.2 SDO error description

If the SDO protocol on a page cannot be processed further, an SDO-Abort telegram is sent (see chap. 3.5.1, p. 21). The error types are coded as follows:

- Error0: Additional error code HB
- Error1: Additional error code LB
- Error2: Error code
- Error3: Error class

| Error class | Error code | Additional code | Description |
|---|---|---|---|
| 0x05 | 0x03 | 0x0000 | The toggle bit is not changed |
| 0x05 | 0x04 | 0x0001 | SDO command specifier invalid or unknown |
| 0x06 | 0x01 | 0x0000 | Access to this object is not supported |
| 0x06 | 0x01 | 0x0001 | Attempt to read a write-only parameter |
| 0x06 | 0x01 | 0x0002 | Attempt to write to a read-only parameter |
| 0x06 | 0x02 | 0x0000 | Object not present in the object dictionary |
| 0x06 | 0x04 | 0x0041 | Object cannot be mapped in a PDO |
| 0x06 | 0x04 | 0x0042 | Number and/or length of the mapped objects exceed the PDO length |
| 0x06 | 0x04 | 0x0043 | General parameter incompatibility |
| 0x06 | 0x04 | 0x0047 | General internal incompatibility error in the device |
| 0x06 | 0x07 | 0x0010 | Data type or parameter length do not match or are unknown |
| 0x06 | 0x07 | 0x0012 | Data types do not match, parameter length too long |
| 0x06 | 0x07 | 0x0013 | Data types do not match, parameter length too short |
| 0x06 | 0x09 | 0x0011 | Subindex not present |
| 0x06 | 0x09 | 0x0030 | General value range error |
| 0x06 | 0x09 | 0x0031 | Value range error: Parameter value too high |
| 0x06 | 0x09 | 0x0032 | Value range error: Parameter value too low |
| 0x06 | 0x09 | 0x0036 | Value range error: Maximum value smaller than minimum value |
| 0x08 | 0x00 | 0x0000 | General SDO error |
| 0x08 | 0x00 | 0x0020 | Cannot be accessed |
| 0x08 | 0x00 | 0x0022 | Cannot be accessed at current device status |

### 3.6 Emergency object (error message)

The emergency object informs other bus participants of errors asynchronously without requiring a query. The emergency object is always 8 bytes in size:

| 11-bit identifier | 8 bytes user data | | | | | | |
|---|---|---|---|---|---|---|---|
| 0x80 (128d) + node ID | Error0(LB) | Error1(HB) | Error-Reg | FE0 (LB) | FE1 (HB) | 0 | 0 | 0 |

Assignment of user data:

- Error0(LB)/Error1(HB): 16-bit error code
- Error-Reg: Error register (contents of object 0x1001, see chap. 5.1, p. 38)
- FE0(LB)/FE1(HB): 16-bit FAULHABER error register (contents of object 0x2320, see Tab. 12)
- Bytes 5 to 7: unused (0)

The error register identifies the error type. The individual error types are bit-coded and are assigned to the respective error codes. The object 0x1001 contains the last value of the error register.

Tab. 9 lists all the errors that have been reported by emergency messages, provided that the respective error is included in the emergency mask for the FAULHABER error register (Tab. 13).

Tab. 9: Emergency error codes

| Emergency message | | FAULHABER error register 0x2320 | | | Error register 0x1001 | |
|---|---|---|---|---|---|---|
| Error Code | Designation | Error mask 0x2321 | Bit | Designation | Bit | Designation |
| 0x0000 | No error (is sent out when an error is no longer present or has been acknowledged) | – | – | – | – | – |
| | | | | | | |
| – | – | – | – | | 0 | Generic error (is set if one of the error bits 1 to 7 is set) |
| | | | | | | |
| 0x3210 | Overvoltage | 0x0004 | 2 | OverVoltageError | 2 | Voltage error |
| 0x3220 | Undervoltage | 0x0008 | 3 | UnderVoltageError | 2 | Voltage error |
| | | | | | | |
| 0x43F0 | Temperature Warning | 0x0010 | 4 | TempWarning | 1 | Current error [a] |
| 0x4310 | Temperature Error | 0x0020 | 5 | TempError | 3 | Temperature error |
| | | | | | | |
| 0x5410 | Output stages | 0x0080 | 7 | IntHWError | 7 | Manufacturer-specific error |
| 0x5530 | EEPROM fault | 0x0400 | 10 | MemError | – | – |
| | | | | | | |
| 0x6100 | Software error | 0x1000 | 12 | CalcError | 7 | Manufacturer-specific error |

| Emergency message | | FAULHABER error register 0x2320 | | | Error register 0x1001 | |
|---|---|---|---|---|---|---|
| Error Code | Designation | Error mask 0x2321 | Bit | Designation | Bit | Designation |
| 0x7200 | Measurement Circuit: Current Measurement | 0x0200 | 9 | CurrentMeasError | 7 | Manufacturer-specific error |
| 0x7300 | Sensor Fault (Encoder) | 0x0040 | 6 | EncoderError | 7 | Manufacturer-specific error |
| 0x7400 | Computation circuit: module fault | 0x0100 | 8 | ModuleError | 7 | Manufacturer-specific error |
| 0x8110 0x8130 0x8140 0x8310 | CAN overrun CAN guarding failed CAN recovered from bus off RS232 overrun | 0x0800 | 11 | ComError | 4 | Communication error |
| 0x84F0 | Deviation error (velocity controller) | 0x0001 | 0 | SpeedDeviationError | 5 | Drive-specific error |
| 0x84FF | Max speed error | 0x2000 | 13 | DynamicError | 7 | Manufacturer-specific error |
| 0x8611 | Following error (position controller) | 0x0002 | 1 | FollowingError | 5 | Drive-specific error |

a) The current controller keeps the motor current below the specified limit at all times. The overcurrent error bit is set if the warning temperature is exceeded. The permissible motor current is then reduced from the peak current value to the continuous current value.

**Example**

An emergency message with the user data assignment in Tab. 10 is sent in the following event:

- In the Error Mask 0x2321, bit 1 (following error) is set under subindex 1 (emergency mask).

- The control deviation corridor set in object 0x6065.00 for the position controller has been exceeded for an extended period as defined by the value set for the error delay time in object 0x6066.00 (see the documentation of the drive functions).

Tab. 10: Example of user data assignment to an emergency message

| 8 bytes user data | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0x11 | 0x86 | 0x20 | 0x02 | 0x00 | 0x00 | 0x00 | 0x00 |

## 3.7 SYNC object

The SYNC object is a message without any user data. The SYNC object is used to trigger synchronous PDOs and at the same time to start processes on various items of equipment.

The identifier of the SYNC objects is set in the object dictionary under the index 0x1005 (by default 0x80).

| 11-bit identifier | 0 bytes user data |
|---|---|
| 0x80 | no user data |

ℹ️ In order that an SYNC object triggers a PDO, the transmission type of the PDO to be triggered must be set accordingly (see Tab. 5).

### 3.7.1 Triggering synchronous PDOs

**Synchronous RxPDO:** The command transmitted with the PDO is not executed until a SYNC object is received. The transmission types 1 to 240 of an RxPDO are identical to transmission type 0.

**Synchronous TxPDO:** The PDO with the current data is not sent until a SYNC object is received.
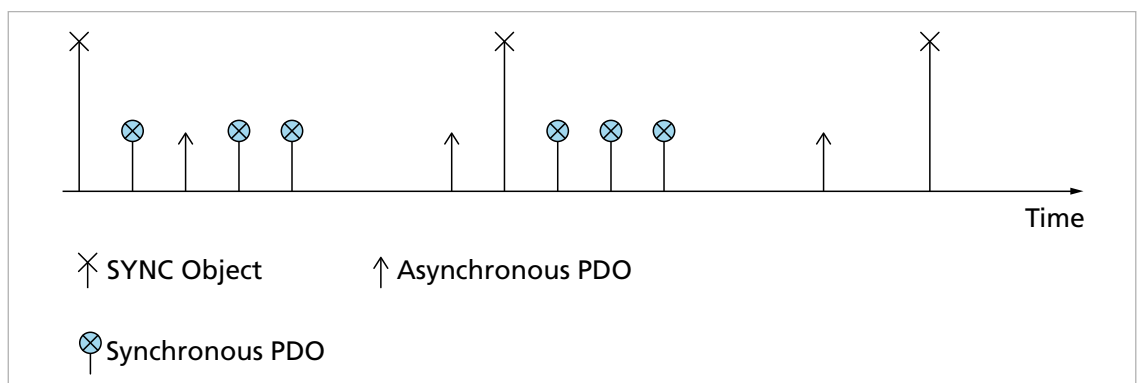


Fig. 6:    TxPDO with SYNC chart

ℹ️ The nodes can also be grouped by transmission types 1-240.

## 3.8    NMT (Network Management)

The network management object governs the CiA 301 state machine of the CANopen device and monitors the network nodes.

After switching on and initializing, the Motion Controller is automatically set to the *Pre-Operational* state. In the *Pre-Operational* state the device can communicate only with NMT messages and via SDOs.
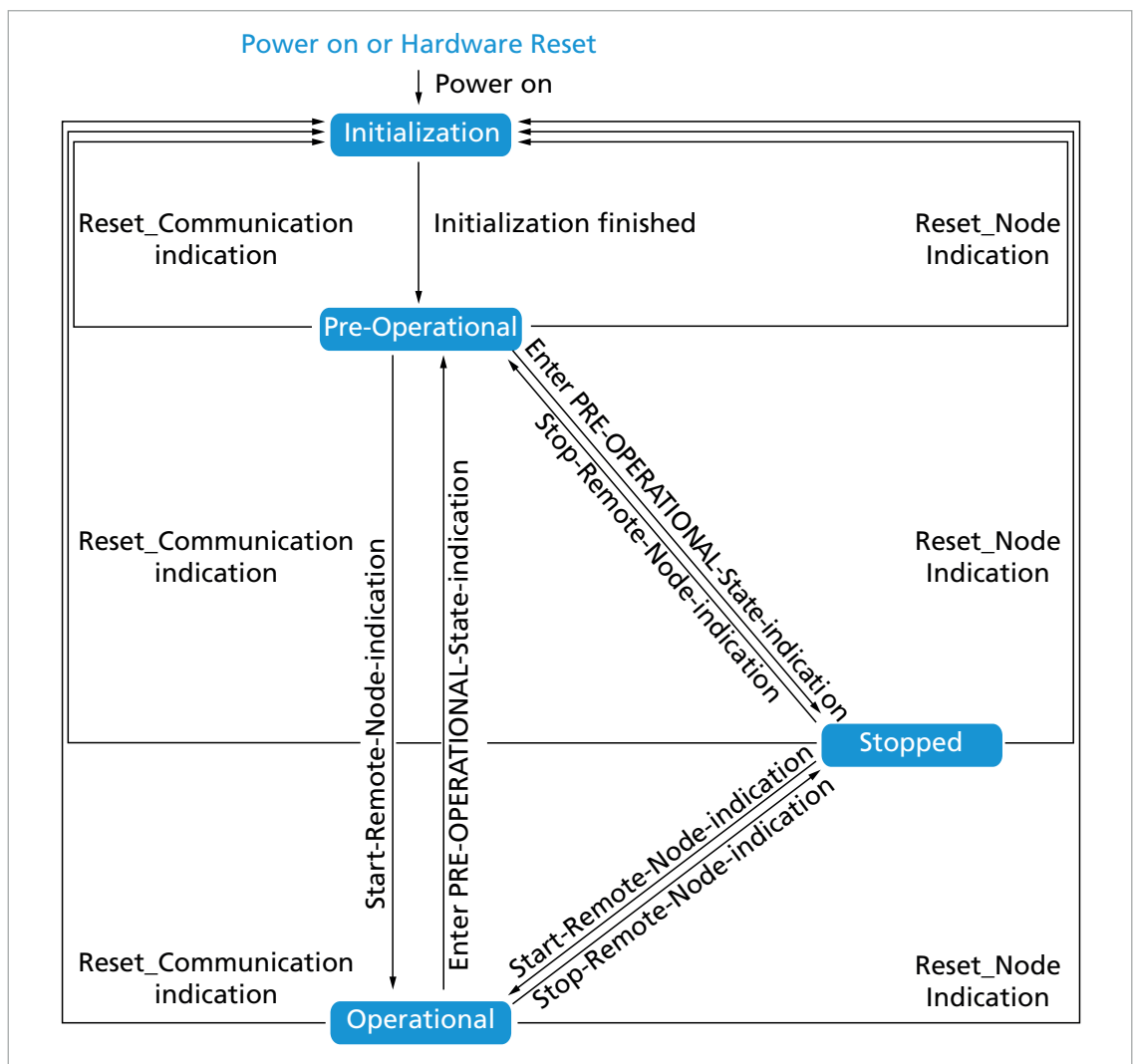


Fig. 7:    CiA 301 state machine

Tab. 11:  NMT changes of status

| Status transition | CS | Meaning |
| --- | --- | --- |
| Power on | – | The initialization state is achieved automatically on switching on. |
| Initialization finished | – | After initialization the device is automatically in the Pre-Operational state, and it sends a boot-up message. |
| Start Remote-Node indication | 0x01 (1d) | This starts the device and enables transmission of PDOs. |
| Enter pre-operational state indication | 0x80 (128d) | Stops the transmission of PDOs, SDOs are still active. |

| Status transition | CS | Meaning |
|---|---|---|
| Stop remote node indication | 0x02 (2d) | The drive is set to the stopped status, SDO and PDO are switched off. |
| Reset Node indication | 0x81 (129d) | Performs a reset. All objects are reset to Power-On standards. |
| Reset Communication indication | 0x82 (130d) | Performs a reset of the communications functions. |

ℹ️ FAULHABER Motion Controllers are equipped with a standard configuration for all objects. Once commissioning is complete the application-specific settings can be saved directly in the device. In most cases no further parametrization is necessary at the system start.

**Starting a CANopen node**
Start Remote-Node:

| 11-bit identifier | 2 bytes user data | |
|---|---|---|
| 0x000 | 0x01 | Node ID |

An entire network can also be started with a CAN message:

Start All Remote-Nodes:

| 11-bit identifier | 2 bytes user data | |
|---|---|---|
| 0x000 | 0x01 | 0x00 |

After the node or the entire network is started the device is in the *Operational* state. The device can now be operated using PDOs.

In the *Stopped* state the device is in an error state and can no longer be operated using PDOs. Under these circumstances, communication with the device is available only by NMT messages.

An NMT message always consists of 2 bytes on the identifier 0x000.

**NMT message**

| 11-bit identifier | 2 bytes user data | |
|---|---|---|
| 0x000 | CS | Node ID |

Assignment of user data:

- CS: Command specifier (see Tab. 11)

- Node ID: Node address (0 = all nodes)

ℹ️ In the event of a serious communications error the Motion Controller switches by default to the *Pre-Operational* NMT status. Different behavior can be set using the object 0x1029.

### 3.8.1 Boot up

Immediately after the initialization phase the Motion Controller sends a boot-up message. A boot-up message signals the end of the initialization phase of a module after it has been switched on. A boot-up message is a CAN message with one data byte (byte 0 = 0x00) on the identifier of the node guarding message (0x700 + node ID).

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

### 3.8.2 Monitoring functions

ℹ️ Only one monitoring function, node guarding or heartbeat can be used at one time.

#### 3.8.2.1 Node guarding

The node guarding object interrogates the current state of the device. To do this, the master sets a remote frame with a request for the guarding identifier of the node to be monitored. The node to be monitored responds with the guarding message which contains the current status of the node and a toggle bit.
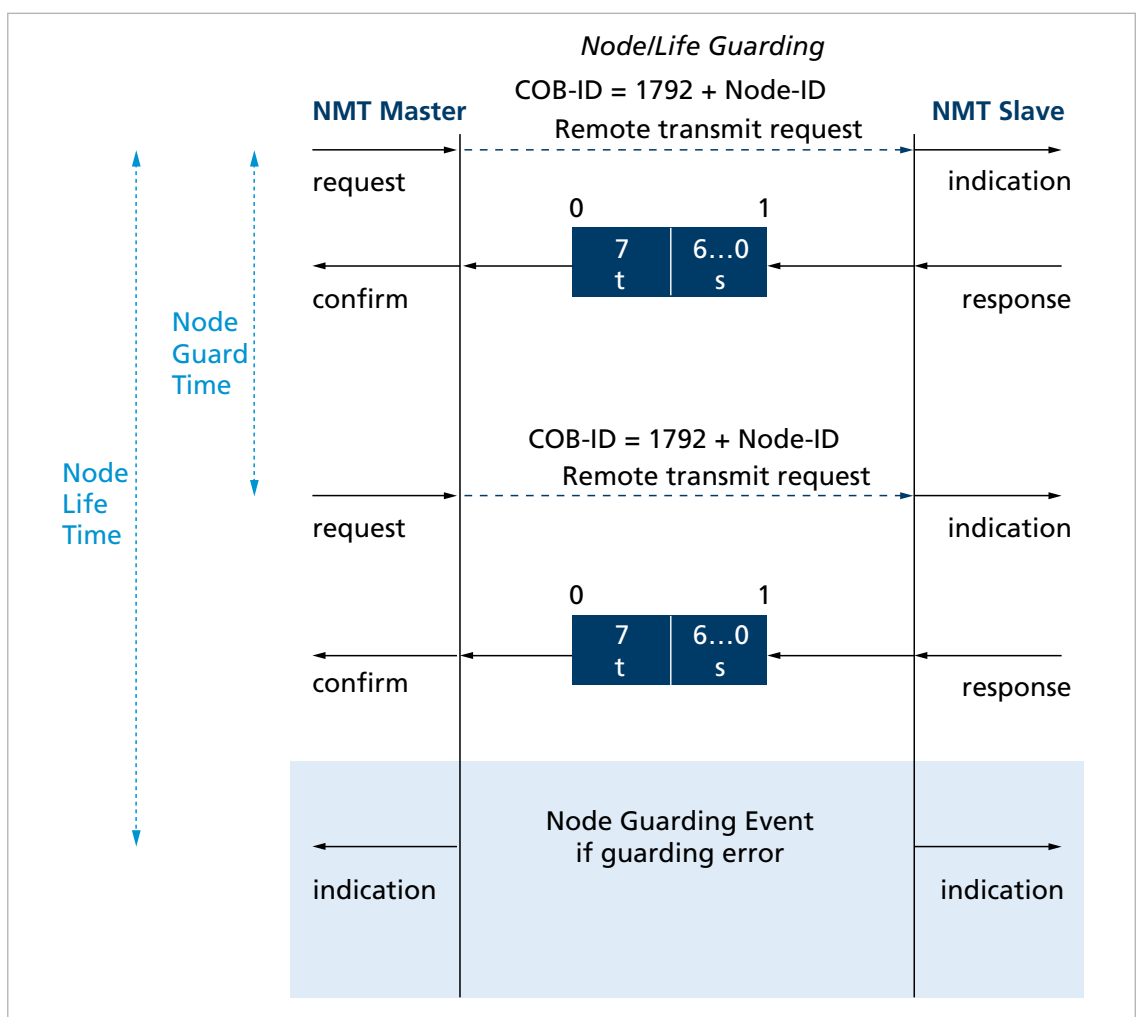
Fig. 8: Chart of the node guarding protocol

*t: Toggle bit*
Initially 0, changes its value at each guarding telegram
*s: Status*
s = 0x04 (4d): Stopped
s = 0x05 (5d): Operational
s = 0x7F (127d): Pre-Operational

If a node life time > 0 is set (objects 0x100C and 0x100D) and no node guarding request is made by the master within the specified life time, a node guarding error is set. The response to a node guarding error is set using the FAULHABER error register (object 0x2321) (see Tab. 14). The default is to send the emergency message 0x8130.

### 3.8.2.2 Heartbeat

The Motion Controller can be set to act both as the heartbeat producer and also as the heartbeat consumer.

■ **Heartbeat producer:** On a cyclical basis the Motion Controller sends out a message which is received by one or more heartbeat consumers in the network.

■ **Heartbeat consumer:** If within the heartbeat consumer time no hearbeat message is received from the heartbeat producer that is being monitored, the Motion Controller responds with the behavior specified in the FAULHABER error register (object 0x2320) (see Tab. 12).
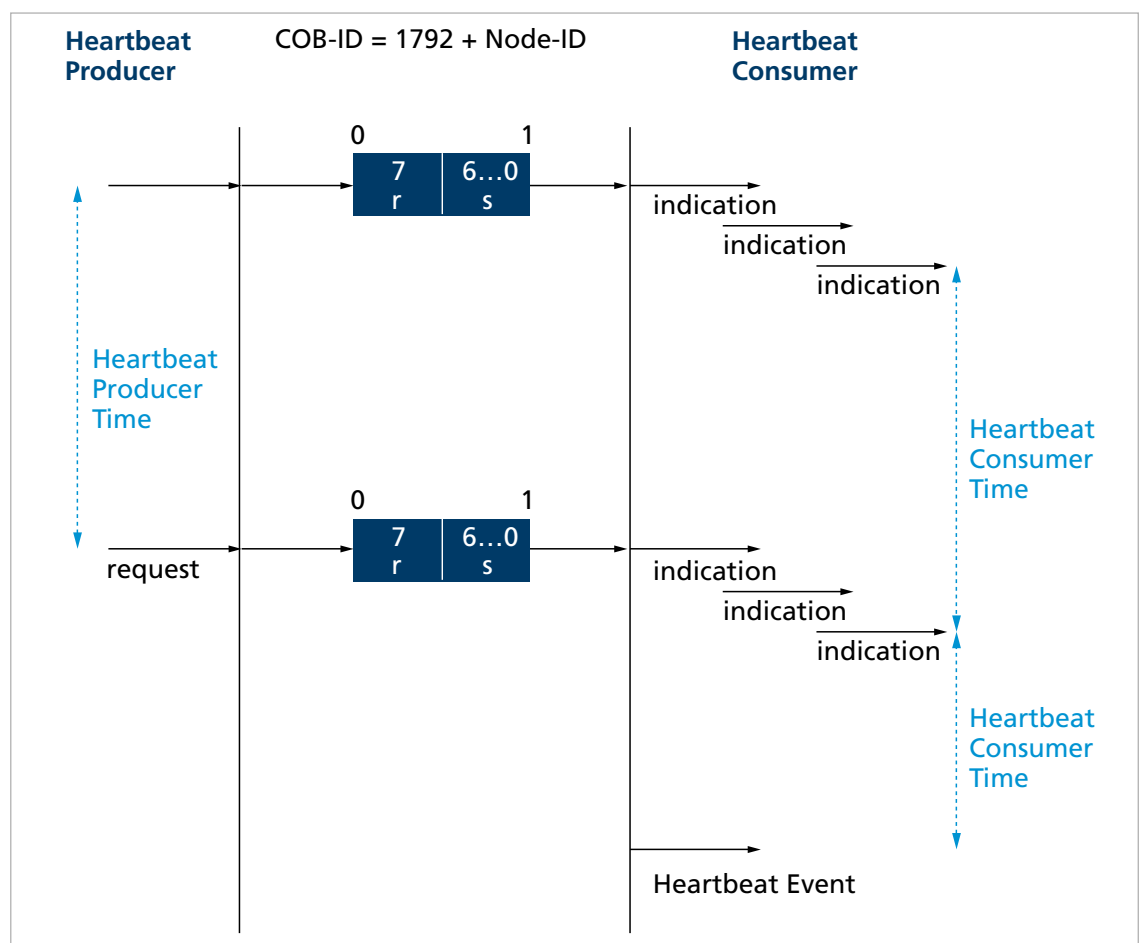


Fig. 9: Chart of the heartbeat protocol

*r: Reserved*
Always 0
*s: Status*
s = 0x00 (0d): Boot-Up
s = 0x04 (4d): Stopped
s = 0x05 (5d): Operational
s = 0x7F (127d): Pre-Operational

### 3.8.3 Settings for the monitoring functions

- Only one of the two monitoring functions (node guarding, heartbeat) can be activated at one time.

- If the producer heartbeat time is > 0 (object 0x1017) the Motion Controller operates as a heartbeat producer. The Motion Controller sends a heartbeat message at the interval for the producer heartbeat time. The node guarding time is set to 0 (see chap. 3.8.2.1, p. 30).

- If the heartbeat is activated, the boot-up message after the switch-on is regarded as the first heartbeat message. Further heartbeats follow at the interval for the producer heartbeat time.

- If in addition to the producer heartbeat time a heartbeat consumer time > 0 is set (object 0x1016.01), the Motion Controller operates as a heartbeat consumer. The settings for the heartbeat producer are ineffective. The node ID of the master to be monitored and the heartbeat consumer time are entered in the object 0x1016.

- The heartbeat consumer time must always be longer than the producer heartbeat time of the master.

- If within the set heartbeat consumer time the Motion Controller receives no heartbeat message from the master, a heartbeat event is triggered. The response to a heartbeat event is determined by the error mask of the FAULHABER error register (object 0x2321) (see Tab. 12). The default is to send the emergency message 0x8130.

- If whilst the heartbeat producer is activated an attempt is made to set a node guarding time, the SDO error 0x08000020 (no access available) is sent.

## 3.9 Entries in the object dictionary

The object dictionary manages the configuration parameters. The object dictionary is divided into three areas. Each object can be referenced by its index and subindex (SDO protocol).

- Communication parameters (index 0x1000 to 0x1FFF) contains communications objects to CiA 301, see chap. 5.1, p. 38)

- Manufacturer-specific area (index 0x2000 to 0x5FFF) contains manufacturer-specific objects, see chap. 5.2, p. 47)

- The standardized device profiles area (0x6000 to 0x9FFF) contains objects supported by the Motion Controller (see the documentation of the drive functions)

## 3.10 Error handling

### 3.10.1 CAN error

**CAN overrun (object lost)**

If messages are lost, the controller sends the emergency message 0x8110. Bit 4 (communication error) is set in the error register and Bit 7 (CAN overrun) is set in the FAULHABER error register. The emergency message is sent out after a delay. Issuing of the emergency message (0x000) does not retract the error. The respective bits in the error register and in the FAULHABER error register are not cleared down.

**CAN in error passive mode**

If the CAN module of the drive is set to the *Error-Passive* state, the emergency message 0x8120 is sent. Bit 4 (communication error) is set in the error register and Bit 6 (CAN in error passive mode) is set in the FAULHABER error register. The emergency message (0x000) is sent and the error retracted once the drive is restored to the *Error-Active* state.

**Recovered from Bus-Off**

If the CAN module of the drive receives a valid message whilst set to the *Bus-Off* state, the emergency message 0x8140 is sent. The emergency message reports that the *Bus-Off* state has been exited. Bit 4 (communication error) is set in the error register and Bit 9 (Recovered from Bus-Off) is set in the FAULHABER error register. This does not retract the error. The respective bits in the error register and in the FAULHABER error register are not cleared down.

> **i** "CAN-Overrun" and "Recovered from bus off" are serious communications errors. The respective bits in the error register and in the FAULHABER error register can be cleared down only by restarting the Motion Controller. Other serious communications errors are:
> - Node guarding timeouts
> - Heartbeat timeouts

### 3.10.2 Device faults

Tab. 12: FAULHABER error register (0x2320)

| Index | Subindex | Name | Type | Attr. | Default value | Meaning |
|-------|----------|------|------|-------|---------------|---------|
| 0x2320 | 0x00 | Fault Register | U16 | ro | – | FAULHABER error register |

The FAULHABER error register contains the most recent errors in bit-coded form. The errors can be masked by selection of the desired types of error via the Error Mask (0x2321) object.

# CANopen protocol description

Tab. 13: Error coding

| Error bit | Error message | Description |
|-----------|---------------|-------------|
| 0x0001 | SpeedDeviationError | Speed deviation too big |
| 0x0002 | FollowingError | Following error |
| 0x0004 | OverVoltageError | Overvoltage detected |
| 0x0008 | UnderVoltageError | Undervoltage detected |
| 0x0010 | TempWarning | Temperature exceeds that at which a warning is output |
| 0x0020 | TempError | Temperature exceeds that at which an error message is output |
| 0x0040 | EncoderError | Error detected at the encoder |
| 0x0080 | IntHWError | Internal hardware error |
| 0x0100 | ModuleError | Error at the external module |
| 0x0200 | CurrentMeasError | Current measurement error |
| 0x0400 | MemError | Memory error (EEPROM) |
| 0x0800 | ComError | Communication error |
| 0x1000 | CalcError | Internal software error |
| 0x2000 | DynamicError | The current velocity is higher than the maximum speed set for the motor. |
| 0x4000 | – | Not used, value = 0 |
| 0x8000 | – | Not used, value = 0 |

All of these errors correspond to an Emergency Error Code. (see chap. 3.6, p. 24).

The error mask describes the handling of internal errors depending on the error coding (see Tab. 13).

Tab. 14: Error Mask (0x2321)

| Index | Subindex | Name | Type | Attr. | Default value | Meaning |
|-------|----------|------|------|-------|---------------|---------|
| 0x2321 | 0x00 | Number of Entries | U8 | ro | 6 | Number of object entries |
| | 0x01 | Emergency Mask | U16 | rw | 0xFFFF | Errors for which an error message is sent |
| | 0x02 | Fault Mask | U16 | rw | 0x0000 | Errors for which the state machine of the drive switches into *Fault Reaction Active* state |
| | 0x03 | Error Out Mask | U16 | rw | 0x0000 | Errors for which the error output pin is set |
| | 0x04 | Disable Voltage Mask | U16 | ro | 0x4024 | Errors which switch off the drive (not configurable) |
| | 0x05 | Disable Voltage User Mask | U16 | rw | 0x0000 | Errors which switch off the drive (config-urable) |
| | 0x06 | Quick Stop Mask | U16 | rw | 0x0000 | Errors for which the state machine of the drive switches into *Quick Stop Active* state |

**Examples:**

- When the fault mask (subindex 2) of object 0x2321 is set to 0x0001 the drive is switched off due to overcurrent and its state machine is set to a *Fault Reaction Active* state.

- When the subindex 3 of object 0x2321 is set to 0, the error output (fault pin) indicates no error. When the subindex 3 of object 0x2321 is set to 0xFFFF, the error output (fault pin) indicates all errors.

# 4 Communication settings

FAULHABER drives are delivered as standard with a pre-set node number 1 and with auto-matically set baud rate detection (AutoBaud).

ℹ️ In network mode the network transmission rate used should be set as the fixed rate.

## 4.1 Setting via the CAN network

For setting via the CAN network the FAULHABER Motion Manager or other configuration tool which supports the LSS protocol (Layer Setting Service and Protocol) to CiA 305 is required.

ℹ️ The FAULHABER Motion Manager must be installed on a PC with a supported CAN interface.

There are two ways of setting the communication parameters:

- An individual drive is connected at the CAN interface of the configuration tool:
  The "LSS Switch Mode Global" without further data allows the drive to be switched to configuration mode, in order to set the node number and Baud rate.

- The drive to be configured is connected via the CAN interface within a network to the configuration tool:
  The "LSS Switch Mode Selective" allows the desired drive to be addressed by inputting the LSS address (Vendor ID, Product code, Revision number, Serial number) and switched to configuration mode, in order to set the node number and Baud rate.

  FAULHABER drives of the MC V3.0 series require the following entries:

  - Vendor ID: 327
  - Product code: 48
  - Revision number: 1.0
  - Serial number: See the product sticker

As well as the setting of the node number and Baud rate, the LSS protocol also supports the reading of the LSS addresses of units that are connected and the reading of the node ID setting.

The identifier 0x7E5 is used (by the master) and 0x7E4 (by the slave) for LSS communication.

After configuration, the Motion Controller saves the set parameters in the EEPROM. They remain available after switching off and on again.

For a detailed description of the LSS protocol please refer to the document CiA 305.

### 4.1.1 Setting the node number

- Node numbers 1 to 127 can be set.
- The node ID 255 (0xFF) marks the node as not configured. After it is switched on, the node is in the LSS-Init status, until a valid node number is assigned. After a valid node number has been assigned to the node, the NMT initialization continues.

### 4.1.2 Setting the baud rate

■ If the automatic baud rate detection (AutoBaud) is active, the drive can be used in a network with any transmission rate in accordance with Tab. 15. The baud rate of the network is detected after no more than 24 telegrams (3 per baud rate) on the bus cable. The drive then sets itself to match the network baud rate.

■ If the automatic baud rate detection is active, telegrams cannot be processed until the baud rate has been detected. If the automatic baud rate detection is active, it takes correspondingly longer to boot up the system.

■ A fixed baud rate in accordance with Tab. 15 can be set by inputting the index 0 to 8.

Tab. 15: Bit timing parameters

| Baud rate | Index |
|-----------|-------|
| 1000 kBit/s | 00 |
| 800 kBit/s | 01 |
| 500 kBit/s | 02 |
| 250 kBit/s | 03 |
| 125 kBit/s | 04 |
| 50 kBit/s | 06 |
| 20 kBit/s | 07 |
| 10 kBit/s | 08 |
| AutoBaud | 09 |

### 4.1.3 Automatic setting of the COB-IDs

If node number 255 (unconfigured CANopen node) is changed to a valid node number, the COB-IDs for the receive and transmit PDOs (RxPDO, TxPDO) and for emergency (EMCY) are automatically set to their default values (see chap. 5.1, p. 38, objects 0x1014.00, 0x1400.01, 0x1401.01, 0x1402.01, 0x1403.01, 0x1800.01, 0x1801.01, 0x1802.01, 0x1803.01).

The configuration must be saved using the save command.

## 4.2 Setting the node number via the object dictionary

As an alternative to the LSS method via the CAN network, the node number can also be set via any interface (CAN, USB, RS232) available on the drive.

The setting is performed by writing the object 0x2400.03 in the object dictionary:

Tab. 16: CAN baud rate index and node number

| Index | Subindex | Name | Type | Attr. | Default value | Meaning |
|---|---|---|---|---|---|---|
| 0x2400 | 0x00 | Number of Entries | U8 | ro | 8 | Number of object entries |
| | 0x01 | CAN Rate | U8 | rw | 9 | Index of the CAN baud rate according to Tab. 15 |
| | 0x03 | Node ID | U8 | rw | 1 | Node number |

The object 0x2400.01 can be used to read the current setting of the baud rate (AutoBaud or fixed baud rate).

A change of the node number via the object 0x2400.03 is acknowledged with the last node number. The changed node number is not loaded until a Save command has been executed for the application parameters followed by a Reset command.

# 5 Parameter description

## 5.1 Communication objects acc. to CiA 301

**Device Type**

| Index | Subindex | Name | Type | Attr. | Default value | Meaning |
|-------|----------|------|------|-------|---------------|---------|
| 0x1000 | 0x00 | Device Type | U32 | ro | 0x00420192 | Indication of the device type |

Contains information on the device type, coded in two 16-bit fields:

- Byte MSB (Most Significant Byte): Additional Information = 0x42 (Servo drive, type specific PDO mapping)
- Byte LSB (Least Significant Byte): Device Profile Number = 0x192 (402d)

**Error Register**

| Index | Subindex | Name | Type | Attr. | Default value | Meaning |
|-------|----------|------|------|-------|---------------|---------|
| 0x1001 | 0x00 | Error Register | U8 | ro | yes | Error register |

The error register contains the last error types that occurred in bit-coded form.

This parameter can be mapped in a PDO.

**Predefined Error Field (error log)**

| Index | Subindex | Name | Type | Attr. | Default value | Meaning |
|-------|----------|------|------|-------|---------------|---------|
| 0x1003 | 0x00 | Number of Errors | U8 | rw | – | Number of saved errors |
| | 0x01–0x08 | Standard Error Field | U32 | ro | – | Error codes that have occurred most recently |

The error log contains the coding of the last errors that occurred.

- Byte MSB: Error Register
- Byte LSB: Error Code

The meaning of the error codes is described in chap. 3.6, p. 24.

Writing a 0 to the subindex 0 clears down the error log.

**COB-ID SYNC**

| Index | Subindex | Name | Type | Attr. | Default value | Meaning |
|-------|----------|------|------|-------|---------------|---------|
| 0x1005 | 0x00 | COB ID SYNC | U32 | rw | 0x80 | CAN object identifier of the SYNC object |

**Manufacturer Device Name**

| Index | Subindex | Name | Type | Attr. | Default value | Meaning |
|-------|----------|------|------|-------|---------------|---------|
| 0x1008 | 0x00 | Manufacturer Device Name | Vis string | const | – | Device name |

Use the segmented SDO protocol to read out the Manufacturer Device Namens.

# Parameter description

**Manufacturer Hardware Version**

| Index | Subindex | Name | Type | Attr. | Default value | Meaning |
|-------|----------|------|------|-------|---------------|---------|
| 0x1009 | 0x00 | Manufacturer Hardware Version | Vis string | const | – | Hardware version |

Use the segmented SDO protocol to read out the Manufacturer Hardware Version.

**Manufacturer Software Version**

| Index | Subindex | Name | Type | Attr. | Default value | Meaning |
|-------|----------|------|------|-------|---------------|---------|
| 0x100A | 0x00 | Manufacturer Software Version | Vis string | const | – | Software version |

The segmented SDO protocol must be used to read the manufacturer's software version.

**Guard Time**

| Index | Subindex | Name | Type | Attr. | Default value | Meaning |
|-------|----------|------|------|-------|---------------|---------|
| 0x100C | 0x00 | Guard Time | U16 | rw | 0 | Monitoring time for node guarding |

Specification of the Guard Time in milliseconds. The value 0 switches node guarding off (chap. 3.8.2.1, p. 30).

**Life Time Factor**

| Index | Subindex | Name | Type | Attr. | Default value | Meaning |
|-------|----------|------|------|-------|---------------|---------|
| 0x100D | 0x00 | Life Time Factor | U8 | rw | 0 | Time factor for node guarding |

The Life Time Factor multiplied by the Guard Time gives the Life Time for the Node Guarding (chap. 3.8, p. 27). The value 0 switches the Node Guarding off.

**Store Parameters**

Tab. 17:  Save parameters

| Index | Subindex | Name | Type | Attr. | Default value | Meaning |
|-------|----------|------|------|-------|---------------|---------|
| 0x1010 | 0x00 | Number of Entries | U8 | ro | 9 | Number of object entries |
| | 0x01 | Save All Parameters | U32 | rw | 1 | Saves all parameters |
| | 0x02 | Save Comm Parameters | U32 | rw | 1 | Save communication parameters (object dictionary entries 0x0000 to 0x1FFF) |
| | 0x03 | Save App Parameters | U32 | rw | 1 | Save application parameters (object dictionary entries 0x2000 to 0x6FFF) |
| | 0x04 | Save App Parameters 1 | U32 | rw | 1 | Save application parameters for immediate changes (set 1) |
| | 0x05 | Save App Parameters  2 | U32 | rw | 1 | Save application parameters for immediate changes (set 2) |

The Store Parameters object saves the configuration parameters into the flash memory. Read access supplies information about the save options. Writing the "Save" signature to the respective subindex initiates the save procedure.

# Parameter description

Tab. 18: Signature "save"

| Signature | ISO 8 859 ("ASCII") | hex |
|---|---|---|
| MSB | e | 65h |
|  | v | 76h |
|  | a | 61h |
| LSB | s | 73h |

⚠️ **NOTICE!**
**The flash memory is designed to accommodate 10,000 write cycles. If this command is executed more than 10,000 times, the correct operation of the flash memory can no longer be guaranteed.**

▶ Avoid performing frequent saves.

▶ After 10,000 save cycles, replace the device.

**Restore Default Parameters**

Tab. 19: Restoring parameters

| Index | Subindex | Name | Type | Attr. | Default value | Meaning |
|---|---|---|---|---|---|---|
| 0x1011 | 0x00 | Number of Entries | U8 | ro | 6 | Number of object entries |
|  | 0x01 | Restore all Default Parameters | U32 | rw | 1 | Restore all factory settings |
|  | 0x02 | Restore Comm Default Parameters | U32 | rw | 1 | Restore all factory settings for communication parameters (0x0000 to 0x1FFF) |
|  | 0x03 | Restore App Default Parameters | U32 | rw | 1 | Restore all factory settings for application parameters (from 0x2000) |
|  | 0x04 | Reload User Parameters | U32 | rw | 1 | Restore the user's last saved application parameters (from 0x2000) |
|  | 0x05 | Reload Application Parameters 1 | U32 | rw | 1 | Application parameter set 1 for immediate changes |
|  | 0x06 | Reload Application Parameters 2 | U32 | rw | 1 | Application parameter set 2 for immediate changes |

The Restore Default Parameters object loads the standard configuration parameters. The standard configuration parameters are either as delivered or as saved last. Read access supplies information about the restore options. Writing the "Load" signature to the respective subindex initiates the restore procedure:

Tab. 20: "Load" signature

| Signature | ISO 8859 ("ASCII") | hex |
|---|---|---|
| MSB | d | 64h |
|  | a | 61h |
|  | o | 6Fh |
| LSB | l | 6Ch |

ℹ️ The delivery state may be loaded only when the output stage is switched off.

# Parameter description

**COB-ID Emergency message**

| Index | Subindex | Name | Type | Attr. | Default value | Meaning |
|---|---|---|---|---|---|---|
| 0x1014 | 0x00 | COB-ID EMCY | U32 | rw | 0x80 + Node-ID | CAN object identifier of the emergency object |

**Consumer Heartbeat Time**

| Index | Subindex | Name | Type | Attr. | Default value | Meaning |
|---|---|---|---|---|---|---|
| 0x1016 | 0x00 | Number of Entries | U8 | ro | 1 | Number of object entries |
| | 0x01 | Consumer Heart-beat Time | U32 | rw | 0 | Heartbeat monitoring time |

- Bits 0 to 15 contain the Consumer Heartbeat Time in milliseconds. If the value is set to 0, the consumer heartbeat function is deactivated (Heartbeat)

- Bits 16 to 23 contain the node number to which the heartbeat message is to be sent (master node ID).

- Bits 24 to 31 are not used (reserved).

**Producer Heartbeat Time**

| Index | Subindex | Name | Type | Attr. | Default value | Meaning |
|---|---|---|---|---|---|---|
| 0x1017 | 0x00 | Producer Heart-beat Time | U16 | rw | 0 | Heartbeat send time interval |

The Producer Heartbeat Time object contains the producer heartbeat time interval in milliseconds. If the value is set to 0, the producer heartbeat function is deactivated (see Settings for the monitoring functions).

**Identity Object**

| Index | Subindex | Name | Type | Attr. | Default value | Meaning |
|---|---|---|---|---|---|---|
| 0x1018 | 0x00 | Number of Entries | U8 | ro | 4 | Number of object entries |
| | 0x01 | Vendor ID | U32 | ro | 327 | Manufacturer's code number (FAULHABER: 327) |
| | 0x02 | Product Code | U32 | ro | 48 | Product code number |
| | 0x03 | Revision Number | U32 | ro | – | Version number |
| | 0x04 | Serial Number | U32 | ro | – | Serial number |

**Error Behaviour**

| Index | Subindex | Name | Type | Attr. | Default value | Meaning |
|---|---|---|---|---|---|---|
| 0x1029 | 0x00 | Number of Entries | U8 | ro | 1 | Number of object entries |
| | 0x01 | Communication Error | U8 | rw | 0 | Behavior in the event of communication errors<br>0 = Pre-operational state<br>1 = No change of state<br>2 = Stopped state |

In the event of a serious communications error the Motion Controller switches to the *Pre-Operational* NMT state. In the event of a serious communication error, subindex 1 allows the behavior to be changed.

# Parameter description

### Server SDO Parameter

| Index | Subindex | Name | Type | Attr. | Default value | Meaning |
|-------|----------|------|------|-------|---------------|---------|
| 0x1200 | 0x00 | Number of Entries | U8 | ro | 2 | Number of object entries |
| | 0x01 | COB ID Client to Server (rx) | U32 | ro | 0x600 + Node-ID | CAN object identifier for RxSDO servers |
| | 0x02 | COB ID Server to Client (tx) | U32 | ro | 0x580 + Node-ID | CAN object identifier for TxSDO servers |

### Receive PDO1 Parameter

| Index | Subindex | Name | Type | Attr. | Default value | Meaning |
|-------|----------|------|------|-------|---------------|---------|
| 0x1400 | 0x00 | Number of Entries | U8 | ro | 2 | Number of object entries |
| | 0x01 | COB ID Used by RxPDO1 | U32 | rw | 0x200 + Node-ID | CAN object identifier for RxPDO1 servers |
| | 0x02 | Transmission Type | U8 | rw | 255 (asynchr.) | PDO transfer type |

### Receive PDO2 Parameter

| Index | Subindex | Name | Type | Attr. | Default value | Meaning |
|-------|----------|------|------|-------|---------------|---------|
| 0x1401 | 0x00 | Number of Entries | U8 | ro | 2 | Number of object entries |
| | 0x01 | COB ID Used by RxPDO2 | U32 | rw | 0x300 + Node-ID | CAN object identifier for RxPDO2 servers |
| | 0x02 | Transmission Type | U8 | rw | 255 (asynchr.) | PDO transfer type |

### Receive PDO3 Parameter

| Index | Subindex | Name | Type | Attr. | Default value | Meaning |
|-------|----------|------|------|-------|---------------|---------|
| 0x1402 | 0x00 | Number of Entries | U8 | ro | 2 | Number of object entries |
| | 0x01 | COB ID Used by RxPDO3 | U32 | rw | 0x400 + Node-ID | CAN object identifier for RxPDO3 servers |
| | 0x02 | Transmission Type | U8 | rw | 255 (asynchr.) | PDO transfer type |

### Receive PDO4 Parameter

| Index | Subindex | Name | Type | Attr. | Default value | Meaning |
|-------|----------|------|------|-------|---------------|---------|
| 0x1403 | 0x00 | Number of Entries | U8 | ro | 2 | Number of object entries |
| | 0x01 | COB ID Used by RxPDO4 | U32 | rw | 0x500 + Node-ID | CAN object identifier for RxPDO4 servers |
| | 0x02 | Transmission Type | U8 | rw | 255 (asynchr.) | PDO transfer type |

### Receive PDO1 Mapping

| Index | Subindex | Name | Type | Attr. | Default value | Meaning |
|---|---|---|---|---|---|---|
| 0x1600 | 0x00 | Number of Mapped Objects | U8 | ro | 1 | Number of mapped objects |
| | 0x01 | RxPDO1 Mapping Entry 1 | U32 | rw | 0x60400010 | Pointer to the 16-bit Controlword (0x6040) |
| | 0x02 | RxPDO1 Mapping Entry 2 | U32 | rw | 0 | |
| | 0x03 | RxPDO1 Mapping Entry 3 | U32 | rw | 0 | |
| | 0x04 | RxPDO1 Mapping Entry 4 | U32 | rw | 0 | |

### Receive PDO2 Mapping

| Index | Subindex | Name | Type | Attr. | Default value | Meaning |
|---|---|---|---|---|---|---|
| 0x1601 | 0x00 | Number of Mapped Objects | U8 | ro | 2 | Number of mapped objects |
| | 0x01 | RxPDO2 Mapping Entry 1 | U32 | rw | 0x60400010 | Pointer to the 16-bit Controlword (0x6040) |
| | 0x02 | RxPDO2 Mapping Entry 2 | U32 | rw | 0x607A0020 | Pointer to the 32-bit Target Position (0x607A) |
| | 0x03 | RxPDO2 Mapping Entry 3 | U32 | rw | 0 | |
| | 0x04 | RxPDO2 Mapping Entry 4 | U32 | rw | 0 | |

### Receive PDO3 Mapping

| Index | Subindex | Name | Type | Attr. | Default value | Meaning |
|---|---|---|---|---|---|---|
| 0x1602 | 0x00 | Number of Mapped Objects | U8 | ro | 2 | Number of mapped objects |
| | 0x01 | RxPDO3 Mapping Entry 1 | U32 | rw | 0x60400010 | Pointer to the 16-bit Controlword (0x6040) |
| | 0x02 | RxPDO3 Mapping Entry 2 | U32 | rw | 0x60FF0020 | Pointer to the 32-bit Target Velocity (0x60FF) |
| | 0x03 | RxPDO3 Mapping Entry 3 | U32 | rw | 0 | |
| | 0x04 | RxPDO3 Mapping Entry 4 | U32 | rw | 0 | |

# Parameter description

### Receive PDO4 Mapping

| Index | Subindex | Name | Type | Attr. | Default value | Meaning |
|---|---|---|---|---|---|---|
| 0x1603 | 0x00 | Number of Mapped Objects | U8 | ro | 2 | Number of mapped objects |
| | 0x01 | RxPDO4 Mapping Entry 1 | U32 | rw | 0x60400010 | Pointer to the 16-bit Controlword (0x6040) |
| | 0x02 | RxPDO4 Mapping Entry 2 | U32 | rw | 0x60710010 | Pointer to the 16-bit Target Torque (0x6071) |
| | 0x03 | RxPDO4 Mapping Entry 3 | U32 | rw | 0 | |
| | 0x04 | RxPDO4 Mapping Entry 4 | U32 | rw | 0 | |

### Transmit PDO1 Parameter

| Index | Subindex | Name | Type | Attr. | Default value | Meaning |
|---|---|---|---|---|---|---|
| 0x1800 | 0x00 | Number of Entries | U8 | ro | 2 | Number of object entries |
| | 0x01 | COB ID Used by TxPDO1 | U32 | rw | 0x180 + Node-ID | CAN object identifier for TxPDO1 |
| | 0x02 | Transmission Type | U8 | rw | 255 (asynchr.) | PDO transfer type |

### Transmit PDO2 Parameter

| Index | Subindex | Name | Type | Attr. | Default value | Meaning |
|---|---|---|---|---|---|---|
| 0x1801 | 0x00 | Number of Entries | U8 | ro | 2 | Number of object entries |
| | 0x01 | COB ID used by TxPDO2 | U32 | rw | 0x280 + Node-ID | CAN object identifier for TxPDO2 |
| | 0x02 | Transmission Type | U8 | rw | 255 (asynchr.) | PDO transfer type |

### Transmit PDO3 Parameter

| Index | Subindex | Name | Type | Attr. | Default value | Meaning |
|---|---|---|---|---|---|---|
| 0x1802 | 0x00 | Number of Entries | U8 | ro | 2 | Number of object entries |
| | 0x01 | COB ID Used by TxPDO3 | U32 | rw | 0x380 + Node-ID | CAN object identifier for TxPDO3 |
| | 0x02 | Transmission Type | U8 | rw | 255 (asynchr.) | PDO transfer type |

### Transmit PDO4 Parameter

| Index | Subindex | Name | Type | Attr. | Default value | Meaning |
|---|---|---|---|---|---|---|
| 0x1803 | 0x00 | Number of Entries | U8 | ro | 2 | Number of object entries |
| | 0x01 | COB ID Used by TxPDO4 | U32 | rw | 0x480 + Node-ID | CAN object identifier for TxPDO4 |
| | 0x02 | Transmission Type | U8 | rw | 255 (asynchr.) | PDO transfer type |

# Parameter description

### Transmit PDO1 Mapping

| Index | Subindex | Name | Type | Attr. | Default value | Meaning |
|---|---|---|---|---|---|---|
| 0x1A00 | 0x00 | Number of Mapped Objects | U8 | rw | 1 | Number of mapped objects |
| | 0x01 | TxPDO1 Mapping Entry 1 | U32 | rw | 0x60410010 | Pointer to the 16-bit Statusword (0x6041) |
| | 0x02 | TxPDO1 Mapping Entry 2 | U32 | rw | 0 | |
| | 0x03 | TxPDO1 Mapping Entry 3 | U32 | rw | 0 | |
| | 0x04 | TxPDO1 Mapping Entry 4 | U32 | rw | 0 | |

### Transmit PDO2 Mapping

| Index | Subindex | Name | Type | Attr. | Default value | Meaning |
|---|---|---|---|---|---|---|
| 0x1A01 | 0x00 | Number of Mapped Objects | U8 | rw | 2 | Number of mapped objects |
| | 0x01 | TxPDO2 Mapping Entry 1 | U32 | rw | 0x60410010 | Pointer to the 16-bit Statusword (0x6041) |
| | 0x02 | TxPDO2 Mapping Entry 2 | U32 | rw | 0x60640020 | Pointer to the 32-bit Position Actual Value (0x6064) |
| | 0x03 | TxPDO2 Mapping Entry 3 | U32 | rw | 0 | |
| | 0x04 | TxPDO2 Mapping Entry 4 | U32 | rw | 0 | |

### Transmit PDO3 Mapping

| Index | Subindex | Name | Type | Attr. | Default value | Meaning |
|---|---|---|---|---|---|---|
| 0x1A02 | 0x00 | Number of Mapped Objects | U8 | rw | 2 | Number of mapped objects |
| | 0x01 | TxPDO3 Mapping Entry 1 | U32 | rw | 0x60410010 | Pointer to the 16-bit Statusword (0x6041) |
| | 0x02 | TxPDO3 Mapping Entry 2 | U32 | rw | 0x606C0020 | Pointer to the 32-bit Velocity Actual Value (0x606C) |
| | 0x03 | TxPDO3 Mapping Entry 3 | U32 | rw | 0 | |
| | 0x04 | TxPDO3 Mapping Entry 4 | U32 | rw | 0 | |

# Parameter description

**Transmit PDO4 Mapping**

| Index | Subindex | Name | Type | Attr. | Default value | Meaning |
|-------|----------|------|------|-------|---------------|---------|
| 0x1A03 | 0x00 | Number of Mapped Objects | U8 | rw | 2 | Number of mapped objects |
| | 0x01 | TxPDO4 Mapping Entry 1 | U32 | rw | 0x60410010 | Pointer to the 32-bit Position Actual Value (0x6064) |
| | 0x02 | TxPDO4 Mapping Entry 2 | U32 | rw | 0x60770010 | Pointer to the 16-bit Torque Actual Value (0x6077) |
| | 0x03 | TxPDO4 Mapping Entry 3 | U32 | rw | 0 | |
| | 0x04 | TxPDO4 Mapping Entry 4 | U32 | rw | 0 | |

# Parameter description

## 5.2    Manufacturer-specific objects

**FAULHABER error register (0x2320)**

| Index | Subindex | Name | Type | Attr. | Default value | Meaning |
|-------|----------|------|------|-------|---------------|---------|
| 0x2320 | 0x00 | Fault Register | U16 | ro | – | FAULHABER error register |

The FAULHABER error register contains the most recent errors in bit-coded form. The errors can be masked by selection of the desired types of error via the Error Mask (0x2321) object.

**Error Mask (0x2321)**

| Index | Subindex | Name | Type | Attr. | Default value | Meaning |
|-------|----------|------|------|-------|---------------|---------|
| 0x2321 | 0x00 | Number of Entries | U8 | ro | 6 | Number of object entries |
| | 0x01 | Emergency Mask | U16 | rw | 0xFFFF | Errors for which an error message is sent |
| | 0x02 | Fault Mask | U16 | rw | 0x0000 | Errors for which the state machine of the drive switches into *Fault Reaction Active* state |
| | 0x03 | Error Out Mask | U16 | rw | 0x0000 | Errors for which the error output pin is set |
| | 0x04 | Disable Voltage Mask | U16 | ro | 0x4024 | Errors which switch off the drive (not configurable) |
| | 0x05 | Disable Voltage User Mask | U16 | rw | 0x0000 | Errors which switch off the drive (config-urable) |
| | 0x06 | Quick Stop Mask | U16 | rw | 0x0000 | Errors for which the state machine of the drive switches into *Quick Stop Active* state |

The states of the drive state machine are described in the documentation for the drive functions.

**Trace Configuration**

| Index | Subindex | Name | Type | Attr. | Default value | Meaning |
|-------|----------|------|------|-------|---------------|---------|
| 0x2370 | 0x00 | Number of Entries | U8 | ro | 10 | Number of object entries |
| | 0x01 | Trigger Source | U32 | wo | 0 | Trigger source |
| | 0x02 | Trigger Thresh-old | S32 | rw | 0 | Trigger threshold |
| | 0x03 | Trigger Delay Offset | S16 | rw | 0 | Trigger delay |
| | 0x04 | Trigger Mode | U16 | rw | 0 | Trigger mode |
| | 0x05 | Buffer Length | U16 | rw | 100 | Buffer length |
| | 0x06 | Sample Time | U8 | rw | 1 | Recording sampling rate 1: in every sampling step |
| | 0x07 | Trace Source of Channel 1 | U32 | wo | 0 | Trace source of channel 1 |
| | 0x08 | Trace Source of Channel 2 | U32 | wo | 0 | Trace source of channel 2 |
| | 0x09 | Trace Source of Channel 3 | U32 | wo | 0 | Trace source of channel 3 |
| | 0x0A | Trace Source of Channel 4 | U32 | wo | 0 | Trace source of channel 4 |

# Parameter description

## Trace Buffer

| Index | Subindex | Name | Type | Attr. | Default value | Meaning |
|---|---|---|---|---|---|---|
| 0x2371 | 0x00 | Number of Entries | U8 | ro | 5 | Number of object entries |
| | 0x01 | Trace State | U16 | ro | 0 | Trigger status |
| | 0x02 | Trace Value of Channel 1 | Vis string | ro | – | Signal buffer, channel 1 |
| | 0x03 | Trace Value of Channel 2 | Vis string | ro | – | Signal buffer, channel 2 |
| | 0x04 | Trace Value of Channel 3 | Vis string | ro | – | Signal buffer, channel 3 |
| | 0x05 | Trace Value of Channel 4 | Vis string | ro | – | Signal buffer, channel 4 |

## CAN baud rate index and node number

| Index | Subindex | Name | Type | Attr. | Default value | Meaning |
|---|---|---|---|---|---|---|
| 0x2400 | 0x00 | Number of Entries | U8 | ro | 8 | Number of object entries |
| | 0x01 | CAN Rate | U8 | rw | 9 | Index of the CAN baud rate according to Tab. 15 |
| | 0x03 | Node ID | U8 | rw | 1 | Node number |
| | 0x04 | Communication Settings | U32 | rw | 0 | Bit mask for communication settings according to Tab. 21 |
| | 0x06 | ComState | U16 | ro | 0 | Bit mask for communication status according to Tab. 22 |

Tab. 21: Meaning of the bits for 0x2400.04 (Communication Settings)

| Bit | Description |
|---|---|
| 0 | Can Mandatory |
| 1 | AsyncDriveStatus |
| 2…31 | Reserved |

Tab. 22: Meaning of the bits for 0x2400.06 (ComState)

| Bit | Description |
|---|---|
| 0…6 | Reserved |
| 7 | Transmit Overflow Signaled |
| 8 | BufferOverflow |
| 9 | GuardingFailed |
| 10 | GuardAgain |
| 11 | BusOffEnd |
| 12 | BusOffStart |
| 13…14 | Reserved |
| 15 | PDOLength |