Application Note
"CANopen Basic Information"

EPOS 24/1, EPOS 24/5, EPOS 70/10
Firmware version 2000h or higher

Introduction

The EPOS is a digital positioning system suitable for DC and EC (brushless) motors with incremental encoders in a modular package. The performance range of these compact positioning controllers ranges from a few watts up to 700 watts.

A variety of operating modes means that all kinds of drive and automation systems can be flexibly assembled using positioning, speed and current regulation. The built-in CANopen interface allows networking to multiple axis drives and online commanding by CAN bus master units.

For fast communication with several EPOS devices, use the CANopen protocol. The individual devices of a network are commanded by a CANopen master.

Objectives

This application note explains the functionality of the CANopen structure and protocol. The configuration process is explained step by step.

Required Tool

maxon motor EPOS Graphical User Interface GUI Version 1.00 or higher
Freely available at http://www.maxonmotor.com category «Service», subdirectory «Downloads», Order number 280937, 280938, 302267, 302287, 275512, 300583

References

maxon motor EPOS Communication Guide
maxon motor EPOS Firmware Specification
Freely available at http://www.maxonmotor.com category «Service», subdirectory «Downloads», Order number 280937, 280938, 302267, 302287, 275512, 300583

CANopen documentation: Specifications ‘DS-301 Version 4.02’ and ‘DSP-402 Version 2.0’
CiA (CAN in Automation e. V.) http://www.can-cia.org.

Network Structure

The CAN interface of the maxon EPOS drives follows the CiA CANopen specification DS-301 communication profile.

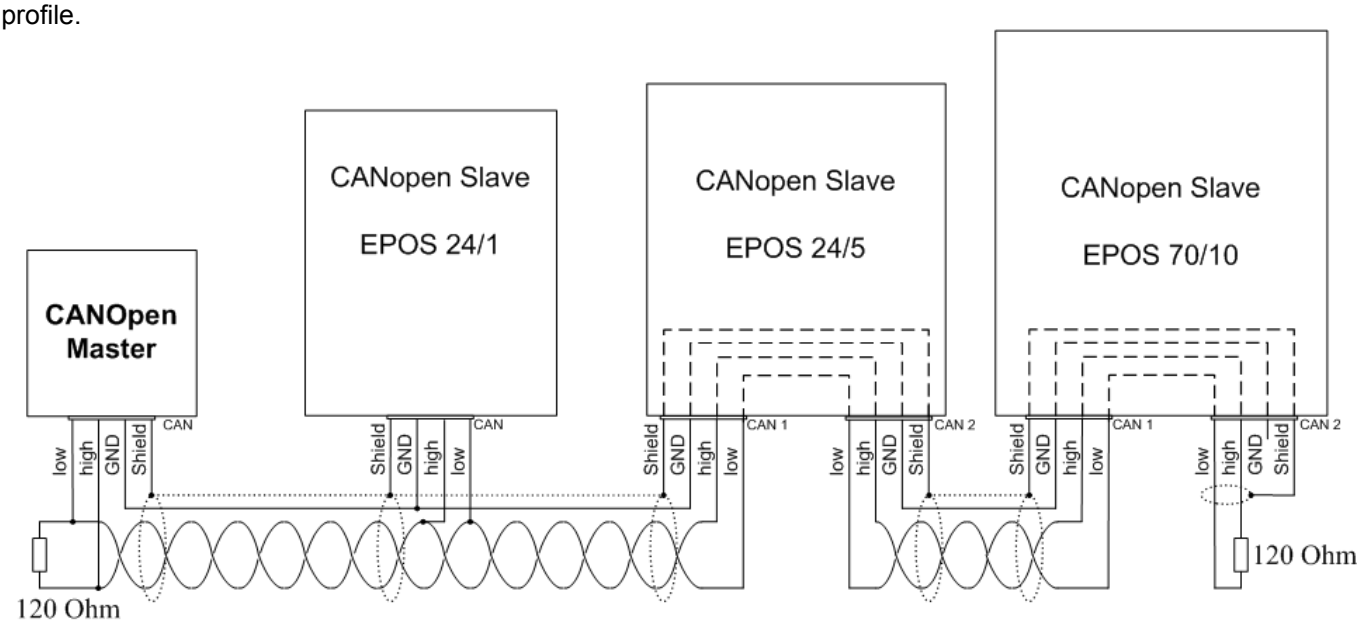

Figure 1: CANopen Network Structure

Configuration

Follow the instructions step by step to set up a correct CAN communication.

Step 1: CANopen Master

Use one of the listed PC CAN interface cards or PLC's. For all of these manufacturers motion control libraries, examples and documentation are available. The latest version may be downloaded freely on internet side of maxon motor ag (http://www.maxonmotor.com).

Recommended PC CAN interface card

Manufacturer / Contact	Supported Products	maxon Motion Control Library
IXXAT ⇨ www.ixxat.de/english/kontakt/distrib.shtml	all offered CAN cards	Windows 32-Bit DLL

Recommended PLC's

Manufacturer / Contact	Supported Products	maxon Motion Control Library
Beckhoff ⇨ www.beckhoff.de	all offered CAN cards	IEC 1131 Beckhoff Library
Siemens ⇨ www.siemens.com/index.jsp Helmholz ⇨ www.helmholz.de	S7-300 with Helmholz CAN300 Master	IEC 1131 Siemens S7 Library
VIPA ⇨ www.vipa.de	VIPA 214-2CM02 CAN-Master	IEC 1131 VIPA Library

Note: All other CAN products of other manufacturers can also be used, however no motion control library is available.

Step 2: CAN Bus Wiring

The two-wire bus line has to be terminated at both ends by a resistor of about 120Ω. The two-wires should be twisted and may be shielded depending on EMC requirements.

Connection EPOS – CAN bus line CiA DS-102

EPOS 24/1 280937, 280938, 302267	**EPOS 24/1, 24/5, 70/10 302287, 275512, 300583**	**CAN 9 pin D-Sub (DIN41652) on PLC or PC CAN interface**
Connector J2 pin 1 “CAN high”	Pin 1 “CAN high”	Pin 7 “CAN_H” high bus line
Connector J2 pin 2 “CAN low”	Pin 2 “CAN low”	Pin 2 “CAN_L” low bus line
Connector J2 pin 5 “CAN GND”	Pin 3 “CAN GND”	Pin 3 “CAN_GND” Ground
CAN shield connect to taphole on EPOS 24/1 housing	Pin 4 “CAN shield”	Pin 5 “CAN_Shield” Cable Shield

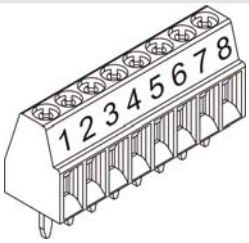

Figure 2: Connector (J2)

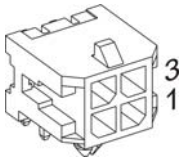

Figure 3: CAN connector Molex Micro-Fit 3.0™ 4 poles (430-25-0400)

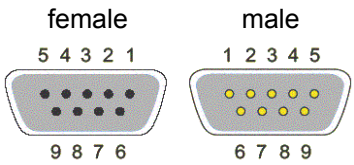

Figure 4: Pin assignment for female and male D-Sub connectors

Step 3: CAN Node ID

For all devices a unique node ID has to be selected.

EPOS 24/1

The CAN-ID (= Node ID) is set at DIP-switch 1 ... 4.
All addresses can be coded from 1 ... 15 using the binary code.

Switch	Binary code	Value
1	2^0	1
2	2^1	2
3	2^2	4
4	2^3	8

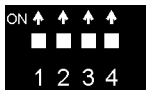

Figure 5: DIP-Switch EPOS 24/1

EPOS 24/5 and EPOS 70/10

The CAN-ID (= Node ID) is set at DIP-switch 1 ... 7.
All addresses can be coded from 1 ... 127 using the binary code.

Switch	Binary code	Value
1	2^0	1
2	2^1	2
3	2^2	4
4	2^3	8
5	2^4	16
6	2^5	32
7	2^6	64

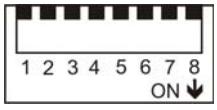

Figure 6: DIP-Switch EPOS 24/5 and 70/10

Step 4: CAN Communication

For the EPOS following CAN bit rates are available:

Object 'CAN Bitrate' (Index 0x2001 SubIndex 0x00)	Bit rate	Max. line length according to CiA DS-102
0	1 MBit/s	25 m
1	800 kBit/s	50 m
2	500 kBit/s	100 m
3	250 kBit/s	250 m
4	125 kBit/s	500 m
5	50 kBit/s	1000 m
6	20 kBit/s	2500 m

All devices on the CAN bus have to use the same bit rate! The maximum bit rate of a CANopen bus depends on the line length. Use the EPOS Graphical User Interface to configure bit rate writing the object 'CAN Bitrate' (Index 0x2001, SubIndex 0x00).

Step 5: Activate Changes

Activate the changes by saving and resetting the EPOS. Execute the menu item 'Save All Parameters' in the menu 'Parameter' of the EPOS Graphical User Interface GUI.

Step 6: Communication Test

Use a CAN monitor program (supported by manufacturer of PC or PLC CAN interface) to check the current wiring and EPOS configuration.

1. Reset all EPOS devices on the bus.
2. At power on the EPOS will send a boot up message.
3. Check that all connected devices send a boot up message (otherwise the EPOS produces a "CAN in Error Passive Mode".
4. Boot up message:
   COB-ID = 0x700 + Node ID
   Data [0] = 0x00

For example the figure below shows the incoming message on CAN bus (EPOS node ID = 1) by a CAN monitor from IXXAT.

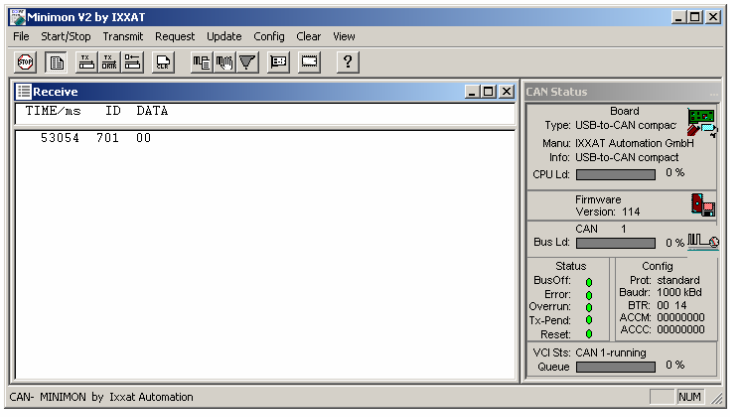

Figure 7: Example boot up message of node 1

SDO Communication

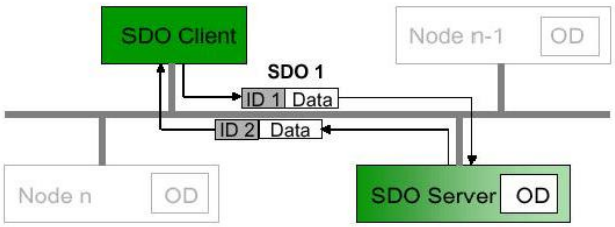

Figure 8: SDO communication

A Service Data Object (SDO) reads from entries or writes to entries of the Object Dictionary. The SDO transport protocol allows transmitting objects of any size. The SDO communication can be used to configure the object of the EPOS.

Two different transfer types are supported. The normal transfer is used for reading or writing objects with a size higher than 4 bytes. This transfer type uses a segmented SDO protocol. This means the transfer is split into different SDO segments (CAN frames). For objects of 4 bytes or less a non-segmented SDO protocol can be used. This transfer is called expedited transfer.

Nearly all objects of the EPOS object dictionary can be read and written using the non-segmented SDO protocol (expedited transfer). Only the data recorder buffer needs to be read using the segmented SDO protocol. For this reason only the non-segmented SDO protocol is explained in this application note. For a description of the segmented protocol (Normal Transfer Type) have a look at the CANopen specification (CiA Draft Standard 301).

Expedited SDO Protocol

Reading Object

	COB-ID	Data [Byte 0]	Data [Byte 1]	Data [Byte 2]	Data [Byte 3]	Data [Byte 4]	Data [Byte 5]	Data [Byte 6]	Data [Byte 7]
Client => Server	0x600 + Node ID		Index LowByte	Index HighByte	Sub-Index	Reserved			

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	1	0	X	X	X	X	X

	COB-ID	Data [Byte 0]	Data [Byte 1]	Data [Byte 2]	Data [Byte 3]	Data [Byte 4]	Data [Byte 5]	Data [Byte 6]	Data [Byte 7]
Server => Client	0x580 + Node ID		Index LowByte	Index HighByte	Sub-Index	Object Byte 0	Object Byte 1	Object Byte 2	Object Byte 3

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	1	0	X	n		e	s

Figure 9: SDO Upload Protocol (Expedited Transfer Type)

Writing Object

	COB-ID	Data [Byte 0]	Data [Byte 1]	Data [Byte 2]	Data [Byte 3]	Data [Byte 4]	Data [Byte 5]	Data [Byte 6]	Data [Byte 7]
Client => Server	0x600 + Node ID		Index LowByte	Index HighByte	Sub-Index	Object Byte 0	Object Byte 1	Object Byte 2	Object Byte 3

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	1	X	n		e	s

	COB-ID	Data [Byte 0]	Data [Byte 1]	Data [Byte 2]	Data [Byte 3]	Data [Byte 4]	Data [Byte 5]	Data [Byte 6]	Data [Byte 7]
Server => Client	0x580 + Node ID		Index LowByte	Index HighByte	Sub-Index	Reserved			

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	1	1	X	X	X	X	X

Figure 10: SDO Download Protocol (Expedited Transfer Type)

Abort SDO Protocol (in case of error)

Server => Client

COB-ID	Data [Byte 0]	Data [Byte 1]	Data [Byte 2]	Data [Byte 3]	Data [Byte 4]	Data [Byte 5]	Data [Byte 6]	Data [Byte 7]
0x580 + Node ID		Index LowByte	Index HighByte	Sub-Index	Abort Code			

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	1	0	X	X	X	X	X

Figure 11: Abort SDO Transfer Protocol

Note: The Abort Codes are described in the document 'EPOS Firmware Specification' in the section 'Communication Errors (Abort Codes)'

Legend: ccs: client command specifier (Bit 7 ... 5)
scs: server command specifier (Bit 7 ... 5)
X: not used; always 0
n: Only valid if e = 1 and s = 1, otherwise 0. If valid it indicates the number of bytes in Data [Byte 4 - 7] that do not contain data. Bytes [8 - n, 7] do not contain segment data.
e: transfer type (0: normal transfer; 1: expedited transfer)
s: size indicator (0: data set size is not indicated; 1: data set size is indicated)

Overview of important command specifier:

Reading Object

Length	*Sending Data [Byte 0]*	*Receiving Data [Byte 0]*
1 Byte	40	4F
2 Byte	40	4B
4 Byte	40	43

Writing Object

Length	*Sending Data [Byte 0]*	*Receiving Data [Byte 0]*
1 Byte	2F	60
2 Byte	2B	60
4 Byte	23	60

SDO Communication Examples

First Example: Read 'Current Regulator P-Gain' (Index 0x60F6 SubIndex 0x01) from node 1

CANopen Sending SDO Frame		
COB-ID	0x601	0x600 + Node ID
Data[0]	0x40	ccs = 2
Data[1]	0xF6	Index LowByte
Data[2]	0x60	Index HighByte
Data[3]	0x01	SubIndex
Data[4]	0x00	reserved
Data[5]	0x00	reserved
Data[6]	0x00	reserved
Data[7]	0x00	reserved

CANopen Receiving SDO Frame		
COB-ID	0x581	0x580 + Node ID
Data[0]	0x4B	scs = 2, n = 2, e = 1, s = 1
Data[1]	0xF6	Index LowByte
Data[2]	0x60	Index HighByte
Data[3]	0x01	SubIndex
Data[4]	0x90	P-Gain LowByte
Data[5]	0x01	P-Gain HighByte
Data[6]	0x00	reserved
Data[7]	0x00	reserved

Current Regulator P-Gain: 0x00000190 = 400

Second Example: Write 'Current Regulator P-Gain' (Index 0x60F6 SubIndex 0x01) to node 1

CANopen Sending SDO Frame		
COB-ID	0x601	0x600 + Node ID
Data[0]	0x2B	ccs = 1, n = 2, e = 1, s = 1
Data[1]	0xF6	Index LowByte
Data[2]	0x60	Index HighByte
Data[3]	0x01	SubIndex
Data[4]	0x12	P-Gain LowByte
Data[5]	0x34	P-Gain HighByte
Data[6]	0x00	reserved
Data[7]	0x00	reserved

CANopen Receiving SDO Frame		
COB-ID	0x581	0x580 + Node ID
Data[0]	0x60	scs = 3
Data[1]	0xF6	Index LowByte
Data[2]	0x60	Index HighByte
Data[3]	0x01	SubIndex
Data[4]	0x00	reserved
Data[5]	0x00	reserved
Data[6]	0x00	reserved
Data[7]	0x00	reserved

PDO Communication

Process Data Objects (PDOs) are used for fast data transmission (real-time data) with a high priority. PDOs are unconfirmed services containing no protocol overhead. Consequently, they represent an extremely fast and flexible method of transmitting data from one node to any number of other nodes. PDOs can contain a maximum of 8 data bytes that can be specifically compiled and confirmed by the user to suit his requirements. Each PDO has a unique identifier and is transmitted by only one node, but it can be received by more than one (producer/consumer communication).

PDO transmissions

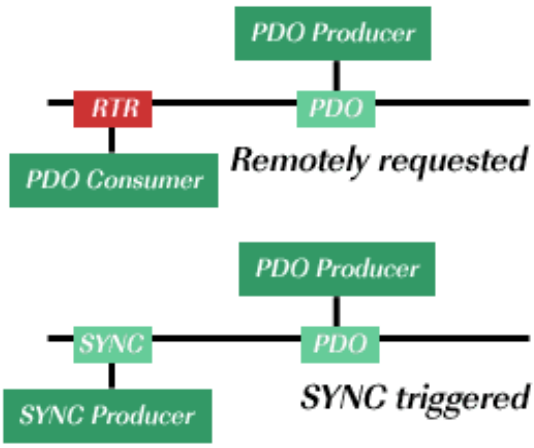

Figure 12: PDO transmissions

PDO transmissions may be driven by remote requests and by the Sync message received:

- *Remotely requested:* Another device may initiate the transmission of an asynchronous PDO by sending a remote transmission request (remote frame).

- *Synchronous transmission:* In order to initiate simultaneous sampling of input values of all nodes, a periodically transmitted Sync message is required. Synchronous transmission of PDOs takes place in cyclic and acyclic transmission mode. Cyclic transmission means that the node waits for the Sync message, after which it sends its measured values. Its PDO transmission type number (1 to 240) indicates the Sync rate it listens to (how many Sync messages the node waits before the next transmission of its values). The EPOS supports only Sync rates of 1.

Acyclically transmitted synchronous PDOs are triggered by a defined application-specific event. The node transmits its values with the next Sync message but will not transmit again until another application-specific event has occurred.

PDO mapping

The default mapping of application objects as well as the supported transmission mode is described in the Object Dictionary for each PDO. PDO identifiers should have high priority to guarantee a short response time. PDO transmission is not confirmed. The PDO mapping defines which application objects are transmitted within a PDO. It describes the sequence and length of the mapped application objects. A device that supports variable mapping of PDOs must support this during the pre-operational state. If dynamic mapping during operational state is supported, the SDO Client is responsible for data consistency.

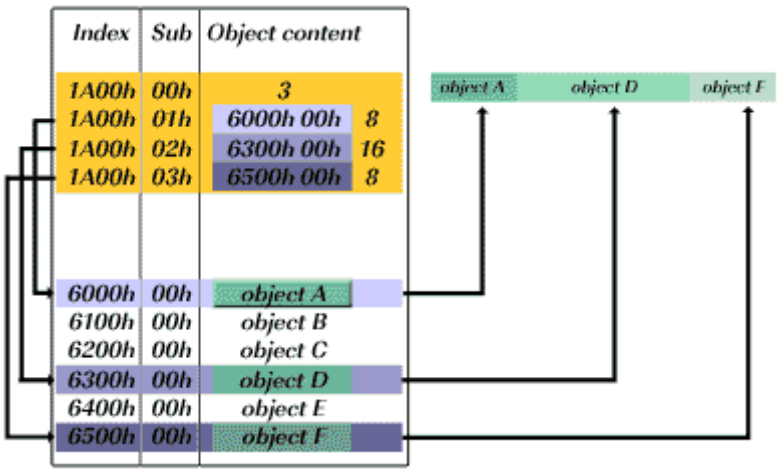

Figure 13: PDO mapping

PDO Configuration

The following section explains step by step how the configuration has to be implemented for PDOs. For all changes in the 'Object Dictionary' described below, use the EPOS Graphical User Interface GUI. For each step an example is noted for 'Receive PDO 1' and 'Node 1'.

<table>
<tr><td>Step 1:
Configure
COB-ID</td><td>The default value of the COB-ID depends on the node ID (Default COB-ID = PDO-Offset + Node ID).

Otherwise the COB-ID can be set in a defined range.

Below a table for all default COB-IDs and ranges of COB-IDs:

<table>
<tr><th>Object</th><th>Default
COB-ID Node 1</th><th>Range
COB-IDs</th></tr>
<tr><td>TxPDO 1</td><td>0x181</td><td>0x181 – 0x1FF</td></tr>
<tr><td>RxPDO 1</td><td>0x201</td><td>0x201 – 0x27F</td></tr>
<tr><td>TxPDO 2</td><td>0x281</td><td>0x281 – 0x2FF</td></tr>
<tr><td>RxPDO 2</td><td>0x301</td><td>0x301 – 0x37F</td></tr>
<tr><td>TxPDO 3</td><td>0x381</td><td>0x381 – 0x3FF</td></tr>
<tr><td>RxPDO 3</td><td>0x401</td><td>0x401 – 0x47F</td></tr>
</table>

All changed COB-IDs can be reset by 'Restore Default PDO COB-IDs'.

Example: ‘COB-ID used by RxPDO 1’ (Index 0x1400, SubIndex 0x00):

Default COB-ID RxPDO 1 = 0x200 + Node ID = 0x201
In Range COB-ID RxPDO 1 = 0x233</td></tr>
<tr><td>Step 2:
Set
Transmission
Type</td><td>Type 0x01: Synchronous Transmission
The data of a synchronous PDO is passed to the application after the occurrence of the SYNC.

Type 0xFF: Asynchronous Transmission
The data of an asynchronous PDO is passed directly to the application.

Example: 'Transmission Type' (Index 0x1400, SubIndex 0x02)

Value = 0x01 or 0xFF</td></tr>
<tr><td>Step 3:
Number of
Mapped
Application
Objects</td><td>Disable the PDO by wiring zero to the object 'Number of Mapped Application Objects in …'

Example: 'Number of Mapped Application Objects in RxPDO 1' (Index 0x1600, SubIndex 0x00)

Value = 0x00</td></tr>
</table>

Step 4: 1st - 4th Mapped Object

Set the value from an object.

Receive **PDO 1** and **PDO 2** mapping objects

1st, 2nd Byte	3rd Byte	4th Byte		
Object Index	Object SubIndex	Object Length in bit	Object Name	Object Value
0x6040	0x00	0x10 (16)	Controlword	0x60400010
0x607A	0x00	0x20 (32)	Target position	0x607A0020
0x60FF	0x00	0x20 (32)	Target velocity	0x60FF0020
0x2078	0x01	0x10 (16)	Digital Output Functionality State	0x20780110
0x2030	0x00	0x10 (16)	Current mode setting value	0x20300010
0x206B	0x00	0x20 (32)	Velocity mode setting value	0x206B0020
0x2062	0x00	0x20 (32)	Position mode setting value	0x20620020

Receive **PDO 3** mapping objects
The mapping of this process data objects are static and can not be changed.

0x6040	0x00	0x10 (16)	Controlword	0x60400010
0x6060	0x00	0x08 (8)	Modes of operation	0x60600008

Transmit **PDO 1** and **PDO 2** mapping objects

1st, 2nd Byte	3rd Byte	4th Byte		
Object Index	Object SubIndex	Object Length in bit	Object Name	Object Value
0x6041	0x00	0x10 (16)	Statusword	0x60410010
0x6064	0x00	0x20 (32)	Position actual value	0x60640020
0x606C	0x00	0x20 (32)	Velocity actual value	0x606C0020
0x2071	0x01	0x10 (16)	Digital Input Functionalities State	0x20710110
0x6078	0x00	0x10 (16)	Current actual value	0x60780010
0x207C	0x01	0x10 (16)	Analog Input 1	0x207C0110
0x207C	0x02	0x10 (16)	Analog Input 2	0x207C0210
0x2020	0x00	0x10 (16)	Encoder counter	0x20200010
0x2021	0x00	0x10 (16)	Encoder counter at index pulse	0x20210010

Transmitting **PDO 3** mapping objects
The mapping of this process data objects are static and can not be changed.

0x6041	0x00	0x10 (16)	Statusword	0x60410010
0x6061	0x00	0x08 (8)	Modes of operation display	0x60610008

Example: '1st - 4th mapped Object in RxPDO 1' (Index 0x1600, SubIndex 0x01 - 0x04)

Step 5: Number of Mapped Application Objects

Enable the PDO by writing the value of the number of objects in 'Number of Mapped Application Objects …'.

PDO 1 and 2: Range 0 … 4
PDO 3: Range 0 … 2

Example: 'Number of Mapped Application Objects in RxPDO 1' (Index 0x1600, SubIndex 0x00)

Step 6: Activate Changes

Activate the changes by saving and resetting the EPOS. Execute the menu item 'Save All Parameters' in the menu 'Parameter' of the EPOS Graphical User Interface GUI.