EPOS

Positioning Controller

Application Note "CANopen Basic Information"

Edition May 2008

EPOS 24/1, EPOS 24/5, EPOS 70/10, MCD EPOS 60W, EPOS2 50/5
Firmware version 2000h or higher

Introduction

The EPOS positioning controller 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 allows all kinds of drive and automation systems to 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.

References and Required Tool

The latest editions of maxon motor documents and tools are freely available at http://www.maxonmotor.com category «Service & Downloads».

Document	Suitable order number for EPOS Positioning Controller
EPOS Communication Guide EPOS Firmware Specification	280937, 302267, 302287, 317270, 275512, 300583
EPOS2 Communication Guide EPOS2 Firmware Specification	347717
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

Tool	
EPOS Studio Version 1.30 or higher	280937, 302267, 302287, 317270, 275512, 347717, 300583

Network Structure

The CAN interface of the maxon EPOS drives follows the CiA CANopen specification DS-301 Version 4.02 Application Layer communication profile and the DSP 402 Version 2.0 Device Profile Drives and Motion Control.

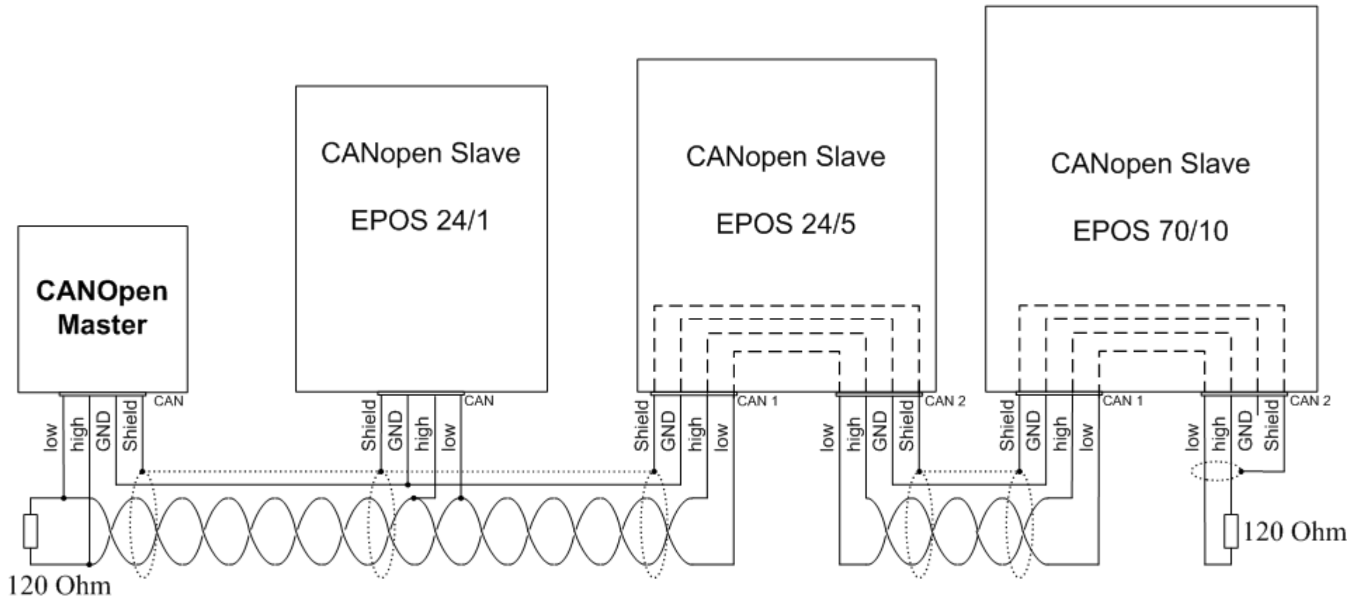


Figure 1: CANopen Network Structure

The CAN-Bus line has to be terminated at both ends with a termination resistor of typically 120 Ω.

Some EPOS Positioning Controller have an internal bus termination which can be switched on with a DIP-Switch:

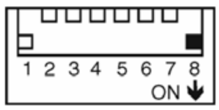
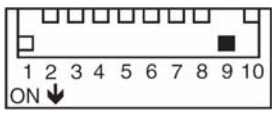
Device	Bus terminated with 120 W	DIP-Switch
EPOS 24/5	DIP-Switch 8 "ON"	
EPOS2 50/5	DIP-Switch 9 "ON":	

Figure 2: DIP-Switch bus termination

Configuration

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

Step 1: CANopen Master

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

Recommended PC CAN interface card

Manufacturer / Contact	Supported Products	maxon Motion Control Library
IXXAT ⇨ www.ixxat.de subdirectory «Contact»	All offered CANopen cards	Windows 32-Bit DLL
Vector ⇨ www.vector-informatik.de	All offered CANopen cards	Windows 32-Bit DLL
National Instruments ⇨ www.ni.com/can	All offered CANopen cards	Windows 32-Bit DLL

Note: The interface driver of the CANopen card must be installed!

Recommended PLC's

Manufacturer / Contact	Supported Products	maxon Motion Control Library
Beckhoff ⇨ www.beckhoff.de	All offered CAN cards	IEC 61131-3 Beckhoff Library
Siemens ⇨ www.siemens.com/index.jsp Helmholz ⇨ www.helmholz.de	S7-300 with Helmholz CAN300 Master	-
VIPA ⇨ www.vipa.de	VIPA 214-2CM02 CAN-Master	IEC 61131-3 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 with a termination resistor of 120 Ω. The two-wires should be twisted and may be shielded depending on EMC requirements.

Connection EPOS Positioning Controller:

EPOS 24/1 (280937, 302267, 317270)	EPOS 24/1 (302287), EPOS 24/5 (275512), EPOS 70/10 (300583), EPOS2 50/5(347717)	MCD EPOS 60W (326343)
Connector J2 pin 1 “CAN high”	Pin 1 “CAN high”	Connector J2 pin 6 “CAN high”
Connector J2 pin 2 “CAN low”	Pin 2 “CAN low”	Connector J2 pin 7 “CAN low”
Connector J2 pin 5 “CAN GND”	Pin 3 “CAN GND	Connector J2 pin 4 “GND”
CAN shield connect to taphole on EPOS 24/1 housing	Pin 4 “CAN shield”	Cable shield soldered on connector housing

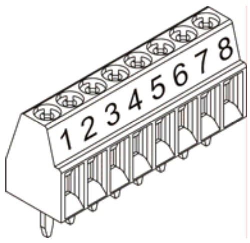


Figure 3: Connector (J2)

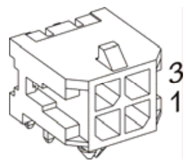


Figure 4: CAN connector Molex Micro-Fit 3.0™ 4 poles (430-25-0400)

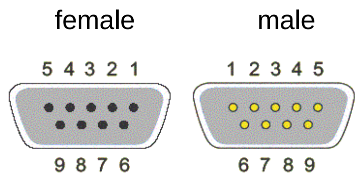


Figure 5: Pin assignment for female and male D-Sub connectors

Connection CAN bus line:

CAN 9 pin D-Sub (DIN41652) on PLC or PC CAN interface	CAN RJ45 on PLC or PC CAN interface
Pin 7 “CAN_H” high bus line	Pin 1 “CAN_H” bus line
Pin 2 “CAN_L” low bus line	Pin 2 “CAN_L” bus line
Pin 3 “CAN_GND” Ground	Pin 3 “CAN_GND” Ground Pin 7 “CAN_GND” Ground
Pin 5 “CAN_Shield” Cable Shield	Pin 6 “CAN_SHLD” Optional CAN Shield

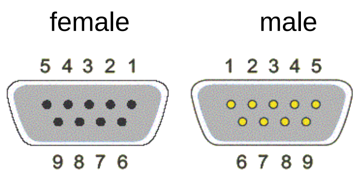


Figure 6: Pin assignment for female and male D-Sub connectors

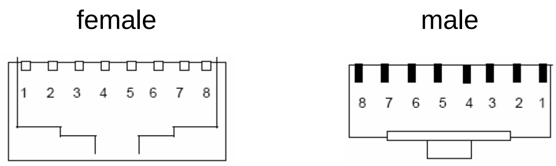


Figure 7: Pin assignment for female and male RJ45 connectors

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 by DIP-Switch 1 ... 4.
All addresses from 1 ... 15 can be coded using the binary code.

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

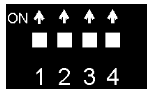


Figure 8: DIP-Switch EPOS 24/1

By setting DIP-Switch address 0 the CAN-ID can be configured by software (changing object 'Node-ID' Index 0x2000 Sub-Index 0x00) ⇨ Range: 1 … 127.

EPOS 24/5 and EPOS 70/10

The CAN-ID (= Node-ID) is set by DIP-Switch 1 ... 7.
All addresses from 1 ... 127 can be coded 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

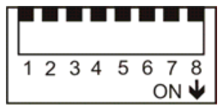


Figure 9: DIP-Switch EPOS 24/5 and 70/10

By setting DIP-Switch address 0 the CAN-ID can be configured by software (changing object 'Node-ID') ⇨ Range: 1 … 127.

MCD EPOS 60W

The CAN-ID (= Node-ID) is detected with Layer setting services (LSS). An exact description is found in the document 'EPOS Firmware Specification'.

EPOS2 50/5

The CAN-ID (= Node-ID) is set by DIP-Switch 1 ... 7.
All addresses from 1 ... 127 can be coded 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

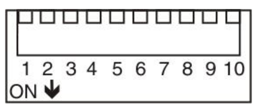


Figure 10: DIP-Switch EPOS2 50/5

By setting DIP-Switch address 0 the CAN-ID can be configured by software (changing object 'Node-ID') ⇨ Range: 1 … 127.

Step 4:
CAN Communication

For the EPOS the following CAN bit rates are available:

Object 'CAN Bit rate' (Index 0x2001 Sub-Index 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 Studio to configure bit rate by writing the object 'CAN Bit rate' (Index 0x2001, Sub-Index 0x00) in the object dictionary.

Step 5:
Activate Changes

Activate the changes by saving and resetting the EPOS.
Execute first menu item 'Save All Parameters', then item 'Reset Node' in the context menu of the selected node in the EPOS Studio.

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.

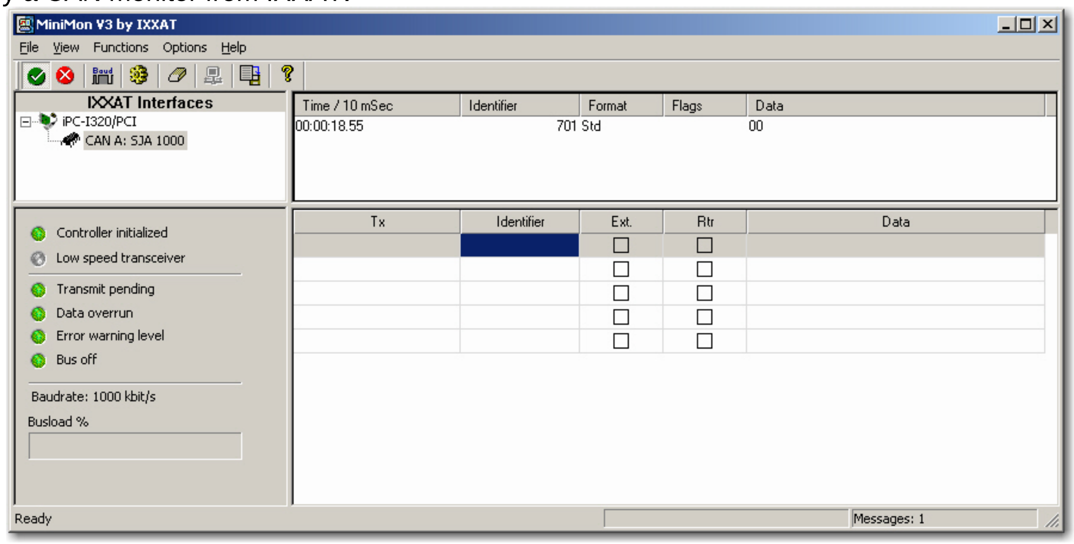


Figure 11: Example boot up message of node 1

SDO Communication

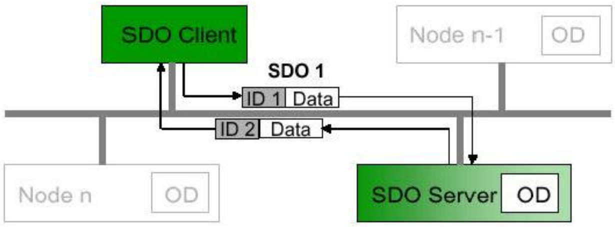


Figure 12: SDO communication

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 Standard 301).

Expedited SDO Protocol

Reading Object

Client => Server

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]
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

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	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 13: SDO Upload Protocol (Expedited Transfer Type)

Writing Object

Client => Server

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]
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

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	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 14: 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 15: 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 ([Byte 0] ⇨Bit 7 … 5):

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 (or 22)	60
2 Byte	2B (or 22)	60
4 Byte	23 (or 22)	60
Not defined	22	60

SDO Communication Examples

Example Read

Read 'Current Regulator P-Gain' (Index 0x60F6 Sub-Index 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	Sub-Index
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	Sub-Index
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

Example Write

Write 'Current Regulator P-Gain' (Index 0x60F6 Sub-Index 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	Sub-Index
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	Sub-Index
Data[4]	0x00	reserved
Data[5]	0x00	reserved
Data[6]	0x00	reserved
Data[7]	0x00	reserved

Current Regulator P-Gain: New Value

Example Abort

Read 'Unknown Object' (Index 0x2000 Sub-Index 0x08) to node 1

CANopen Sending SDO Frame		
COB-ID	0x601	0x600 + Node-ID
Data[0]	0x40	ccs = 2
Data[1]	0x00	Index LowByte
Data[2]	0x20	Index HighByte
Data[3]	0x08	Sub-Index
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]	0x80	scs = 3
Data[1]	0x00	Index LowByte
Data[2]	0x20	Index HighByte
Data[3]	0x08	Sub-Index
Data[4]	0x11	Abort Code [Byte 0]
Data[5]	0x00	Abort Code [Byte 1]
Data[6]	0x09	Abort Code [Byte 2]
Data[7]	0x06	Abort Code [Byte 3]

Abort code: 0x06090011

=> The last read or write command had a wrong object sub index

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).
The CANopen network management is node-oriented and follows a master/slave structure. It requires one device in the network, which fulfils the function of the **NMT (Network Management)** Master. The other nodes are NMT Slaves.

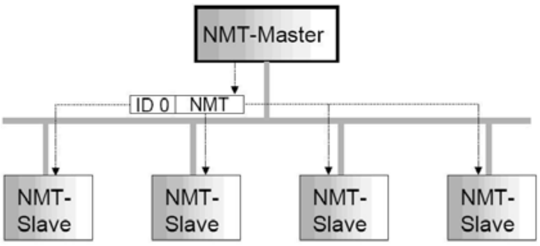


Figure 16 : Network Management (NMT)

The CANopen NMT Slave devices implement a state machine, which brings every device in **Pre-Operational** state automatically after power-on and internal initialisation. In this state the node may be configured and parameterised via SDO (e.g. using a configuration tool), **no PDO communication is allowed.**

⇨To switch from Pre-Operational to Operational State, you have to send the 'Start Remote Node Protocol':

Start Remote Node Protocol

COB-ID	**CS** (Byte 0)	**Node-ID** (Byte 1)	**Functionality**
0	0x01	0 (all)	All EPOS (all CANopen nodes) will enter the Operational NMT State
0	0x01	n	The EPOS (or CANopen node) with the Node-ID n will enter the Operational NMT State

⇨To switch from Operational to Pre-Operational State, you have to send the 'Enter Pre-Operational Protocol':

Enter Pre-Operational Protocol

COB-ID	**CS** (Byte 0)	**Node-ID** (Byte 1)	**Functionality**
0	0x80	0 (all)	All EPOS (all CANopen nodes) will enter the Pre-Operational NMT State
0	0x80	n	The EPOS (or CANopen node) with the Node-ID n will enter the Pre-Operational NMT State

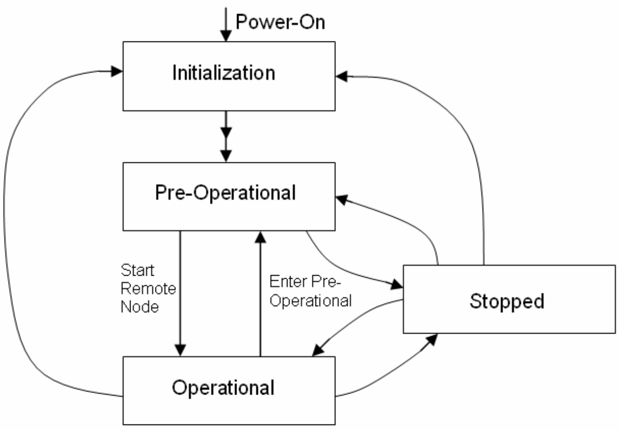


Figure 17 : NMT Slave State Diagram

Note: More information about NMT Services can be found in the document 'Communication Guide'.

PDO Transmissions

PDO transmissions may be driven by remote requests, event triggered 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).

- *Event Triggered (only Transmit PDOs):*
  An event of a mapped Object (e.g. velocity changed) will cause the transmission of this TxPDO. Sub-Index 3h of the object 'Transmit PDO X Parameter' contains the inhibit time. This time is a minimum interval for PDO transmission. The value is defined as multiple of 100 us.

- *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.**

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.

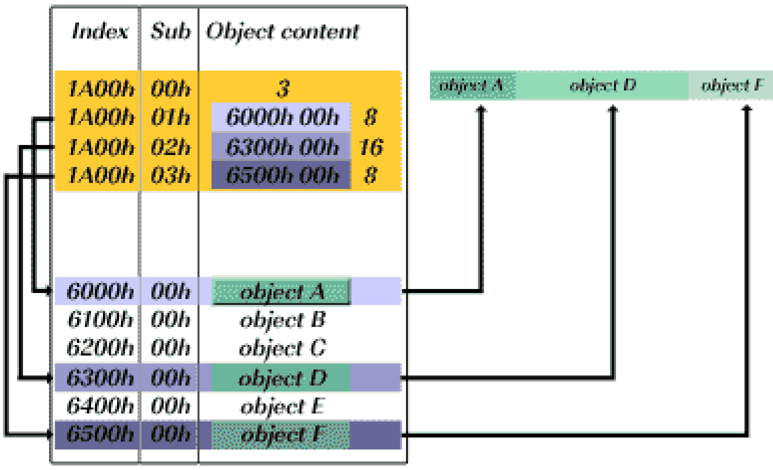


Figure 18: PDO mapping

PDO Configuration

For PDO Configuration you have to be in the Pre-Operational state!

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 Studio. For each step an example is noted for 'Receive PDO 1' and 'Node 1'.

Step 1: Configure COB-ID

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:

Object	Index	Sub-Index	Default COB-ID Node 1
TxPDO 1	0x1800	0x01	0x181
TxPDO 2	0x1801	0x01	0x281
TxPDO 3	0x1802	0x01	0x381
TxPDO 4	0x1803	0x01	0x481
RxPDO 1	0x1400	0x01	0x201
RxPDO 2	0x1401	0x01	0x301
RxPDO 3	0x1402	0x01	0x401
RxPDO 4	0x1403	0x01	0x501

All changed COB-IDs can be reset by 'Restore Default PDO COB-IDs' in the context menu on *'Object Dictionary'* view of the EPOS Studio.

Example: Object ⇨ 'COB-ID used by RxPDO 1' (Index 0x1400, Sub-Index 0x01):

Default COB ID RxPDO 1 = 0x200 + Node-ID = 0x201
In Range COB ID RxPDO 1 = 0x233

Step 2: Set Transmission Type

Type 0x01	TxPDOs	The data is sampled and transmitted after the occurrence of the SYNC.
	RxPDOs	The data is passed to the EPOS and transmitted after the occurrence of the SYNC.
Type 0xFD	TxPDOs	The data is sampled and transmitted after the occurrence of a remote transmission request (RTR).
Type 0xFF	TxPDOs	The data is sampled and transmitted after the occurrence of a remote transmission request or an internal event (value changed).
	RxPDOs	The data is passed directly to the EPOS application

Example: Object ⇨ 'Transmission Type' (Index 0x1400, Sub-Index 0x02)

Type = 0xFF

Step 3: Number of Mapped Application Objects

Disable the PDO by wiring zero to the object 'Number of Mapped Application Objects in …'

Example: Object ⇨ 'Number of Mapped Application Objects in RxPDO 1' (Index 0x1600, Sub-Index 0x00)

Value = 0x00

<table>
<tr><td>Step 4:
Mapping Objects</td><td>Set the value from an object.

Example: Object1 ⇨ ‘1st Mapped Object in RxPDO 1’ (Index 0x1600, Sub-Index 0x01)
Object2 ⇨ ‘2nd Mapped Object in RxPDO 1’ (Index 0x1600, Sub-Index 0x02)
Object3 ⇨ ‘3rd Mapped Object in RxPDO 1’ (Index 0x1600, Sub-Index 0x03)

<table>
<tr><td>RxPDO1</td><td>No.</td><td>Mapped Object</td></tr>
<tr><td></td><td>1.</td><td>Object1 = 0x60400010 ⇨ Controlword (16bit)</td></tr>
<tr><td></td><td>2.</td><td>Object2 = 0x607A0020 ⇨ Target Position (32bit)</td></tr>
<tr><td></td><td>3.</td><td>Object3 = 0x60FB0210 ⇨ Position Regulator I-Gain (16bit)</td></tr>
</table>

All PDOs are dynamic. 8 Bytes (64bit) can be mapped in a PDO (max. 8 Objects).

Note: Appendix PDO Object!

Note: All mappable objects are listened in the associated document “Firmware Specification” (see tables Receive/Transmit PDO mapping objects).</td></tr>
<tr><td>Step 5:
Number of Mapped Application Objects</td><td>Enable the PDO by writing the value of the number of objects in 'Number of Mapped Application Objects in …'.

Example: Object ⇨ 'Number of Mapped Application Objects in RxPDO 1' (Index 0x1600, Sub-Index 0x00)

Value = 0x03</td></tr>
<tr><td>Step 6:
Activate Changes</td><td>The changes are directly activated.
Execute the menu item ‘Save All Parameter’ in the context menu from the used node (EPOS Studio – Navigation Window → Workspace or Communication) or in the context menu in the view ’Object Dictionary’.</td></tr>
</table>

Node Guarding Protocol

To detect absent devices (e.g. because of bus-off) which do not transmit PDOs regularly. The NMT Master can manage a database, where besides other information the expected states of all connected devices are recorded, which is known as Node Guarding. With cyclic Node Guarding the NMT Master regularly polls its NMT Slaves. To detect the absence of the NMT Master, the slaves test internally, whether the Node Guarding is taking place in the defined time interval (Life Guarding). The Node Guarding is initiated by the NMT Master in Pre-Operational state of the slave by transmitting a Remote Frame. Node Guarding is also activated in the Stopped State active.

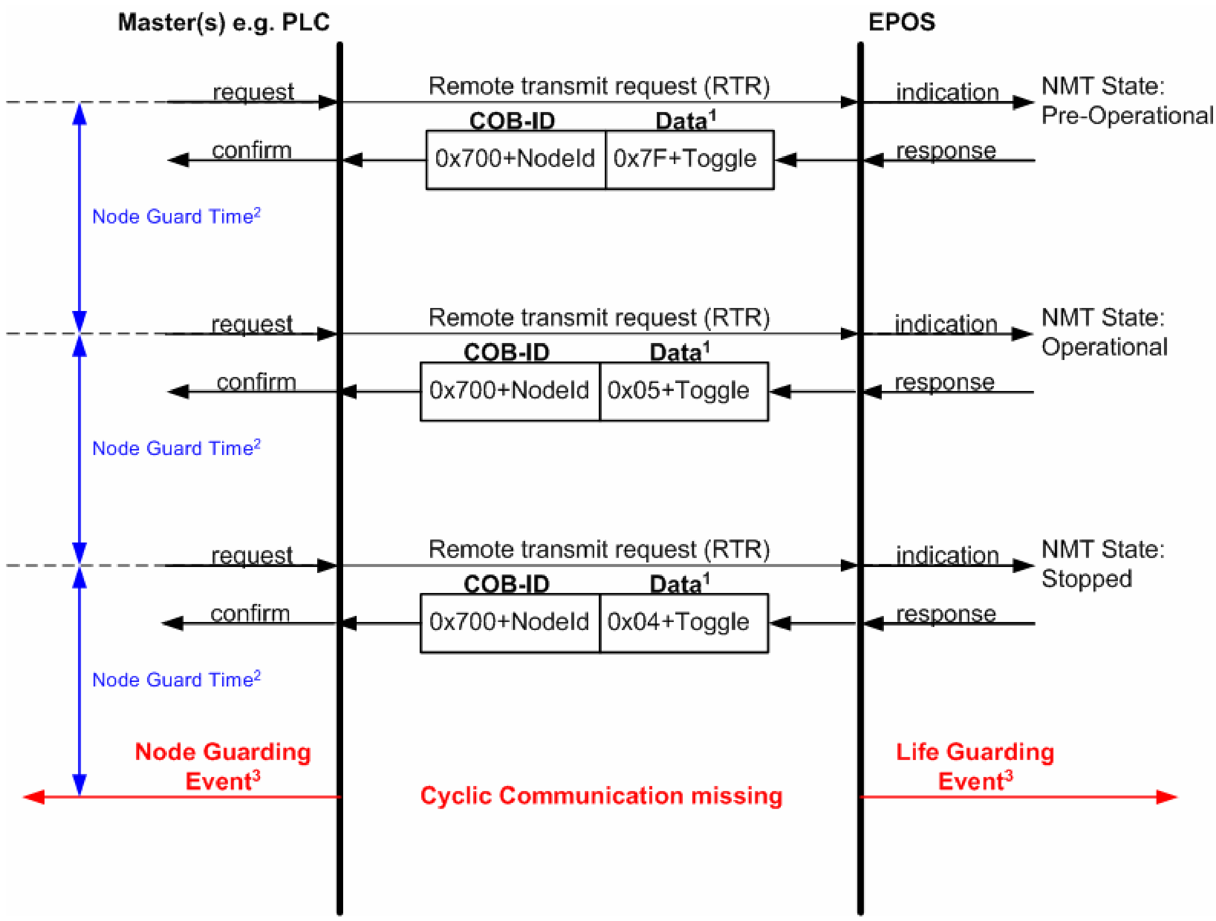


Figure 16: Node Guarding Protocol Timing Diagram

Notes:

[1] **Data Field**

The data field holds the NMT State. Each time the value of toggle changes between 0x00 and 0x80. Therefore the following values for the data field are possible:

Value	*Toggle*	*EPOS NMT State*
0x04	not set	Stopped
0x84	set	Stopped
0x05	not set	Operational
0x85	set	Operational
0x7F	not set	Pre-Operational
0xFF	set	Pre-Operational

[2] **Node Guard Time** is calculated by the following Objects: NodeGuardTime = GuardTime * LifeTimeFactor

[3] **Node / Life Guarding Event**

In case the Remote Transmit Request (RTR) is missed by the EPOS it will change it's device state to error (Node Guarding Error).

In case the answer is missed by the Master System, it should react conveniently with the Node Guarding Event.

Heartbeat Protocol

The Heartbeat Protocol has a higher priority than the Node Guarding Protocol. If both are enabled, only the Heartbeat Protocol is supported. The EPOS transmits a heartbeat message cyclically if the Heartbeat Protocol is enabled (Heartbeat Producer Time 0 = Disabled, Heartbeat Producer Time greater than 0 = enabled). The Heartbeat Consumer guards the reception of the Heartbeat within the Heartbeat Consumer Time. If the Heartbeat Producer Time is configured on the EPOS it starts immediately with the Heartbeat Protocol.

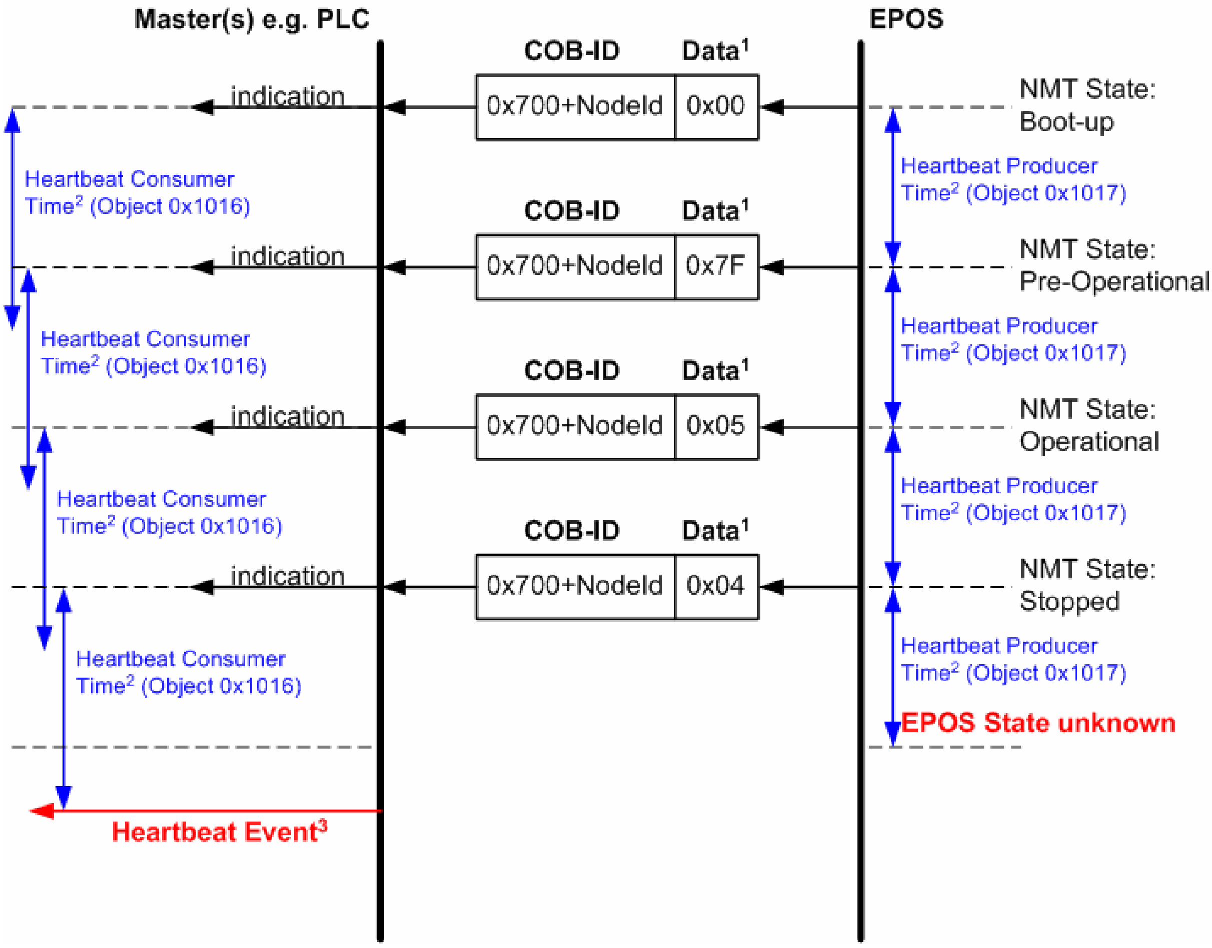


Figure 17: Heartbeat Protocol Timing Diagram

Notes:

[1] **Data Field**
The Data Field holds the NMT State:

Value	*EPOS NMT State*
0x00	Boot-Up
0x04	Stopped
0x05	Operational
0x7F	Pre-Operational

[2] **Heartbeat Producer- and Heartbeat Consumer Time**
The Heartbeat Consumer Time has to be longer than the Heartbeat Producer Time because of generation-, sending- and indication time (⇨ Heartbeat Consumer Time ≥ Heartbeat Producer Time + 5ms). Each indication of the Master resets the Heartbeat Consumer Time.

[3] **Heartbeat Event**
If the EPOS is in an unknown state (e.g. there is no longer a supply voltage on the device) the Heartbeat Protocol can't be sent to the Master. The Master recognizes this after the Heartbeat Consumer Time and generates a Heartbeat Event.