EPOS

Positioning Controller

Documentation

Communication Guide

maxon document #573055-05

maxon motor ag Brünigstrasse 220 P.O. Box 263 CH-6072 Sachseln Tel.: 041/666 15 00 Fax.: 041/666 16 50 www.maxonmotor.com

1 Table of contents

2 Table of figures

3 Table of tables

4 Introduction

This documentation “Communication Guide” provides the communication interfaces details of the EPOS positioning controllers. It contains descriptions of the serial RS232 and the CAN interface.

The maxon motor EPOS are small-sized full digital smart motion controller. Due to the flexible and high efficient power stage the EPOS drives brushed DC motors with digital encoder as well as brushless EC motors with digital Hall sensors and encoder.
The sinusoidal current commutation by space vector control offers to drive brushless EC motors with minimal torque ripple and low noise. The integrated position-, velocity- and current control functionality allows sophisticated positioning applications.
It is specially designed being commanded and controlled as a slave node in the CANopen network. In addition the unit can be operated through any RS-232 communication port.

The latest edition of these “Communication Guide”, additional documentation and software to the EPOS positioning controller may also be found on the internet in http://www.maxonmotor.com category <Service&Downloads>.

5 How to use this guide

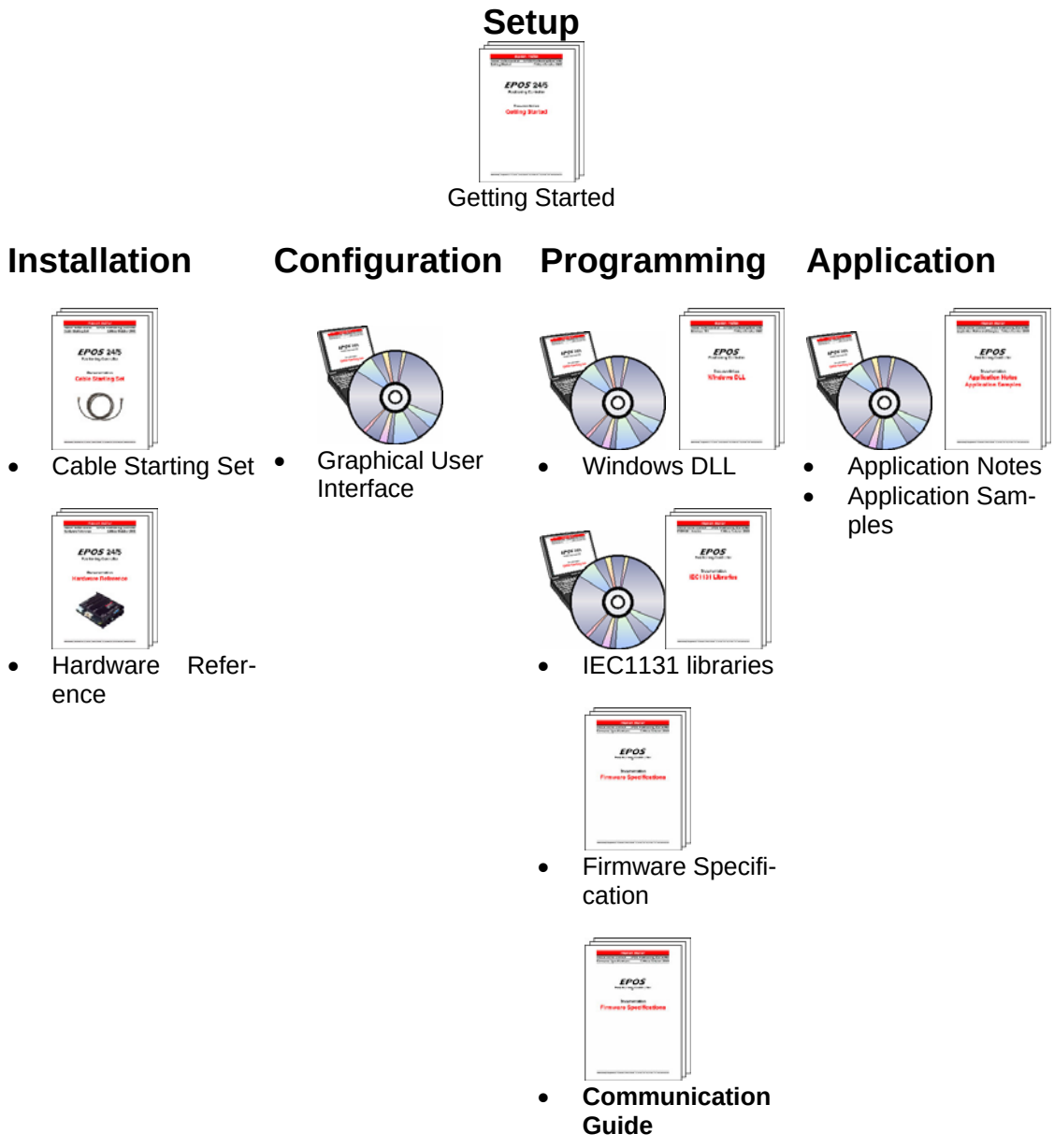


Figure 1: EPOS documentation hierarchy

6 Serial EIA-RS232 Communication

The serial RS232 communication protocol was developed for transmitting and receiving data over the RS232 serial port of an EPOS. Its principal task is to transmit data from a master (Personal Computer or any other central processing unit) to a single slave. The protocol is defined for a point-to-point communication based on the EIA-RS232 standard.
The protocol can be used to implement the command set defined for the EPOS. For a high degree of reliability in an electrically noisy environment it is designed with a checksum.

6.1 Physical Layer

6.1.1 Electrical Standard

The EPOS communication protocol uses the RS232 standard for transmitting data over a three wires cable, for the signals TxD, RxD and GND.
The RS232 standard can be used only for a point-to-point communication between a master and a single EPOS slave. The standard uses negative, bipolar logic in which a negative voltage signal represents a logic '1', and positive voltage represents a logic '0'. Voltages of –3V to –25V with respect to signal ground (GND) are considered logic '1', whereas voltages of +3V to 25V are considered logic '0'.

6.1.2 Medium

For the physical connection a 3 wire cable is required. It is recommended to install a shielded and twisted pair cable in order to have a good performance even in an electrically noisy environment. Depending on the bit rate used the cable length can range from 3 meters up to 15 meters. However we do not recommend RS232 cables longer than 5 meters.

6.2 Data Link Layer

6.2.1 Data Format

Data is transmitted in an asynchronous way, that means each byte of data is transmitted individually with its own start and stop bit.

The format is:

1 Start bit, 8 Data bits, No parity, 1 Stop bit

Most serial communication chips (SCI, UART) can generate such data format.

6.2.2 Frame Structure

The data bytes are transmitted sequentially in frames. A frame is made of a header, a variably long data field and a 16-bit long cyclic redundancy check (CRC) for data integrity checking.

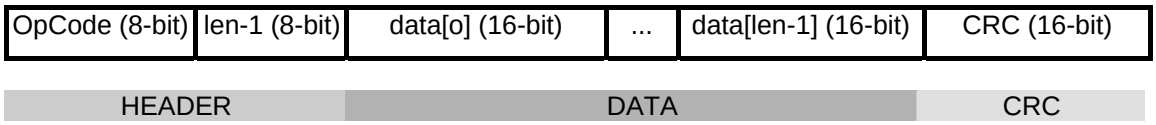


Figure 2: Frame structure

HEADER: The header consists of 2 bytes. The first field determines the type of data frame to be sent or received. The second field contains the length of the data fields.

OpCode: Operation command to be sent to the slave. See the documentation of the command set (EPOS Command Reference).

len-1: 'Len' represents the number of words (16-bit value) in the data fields. The field 'Len-1' contains the number of words minus one. The smallest value in this field is zero, which represents a data length of one word. The data block must contain at least 1 word.

Examples:		
	1 word	len-1 = 0
	2 words	len-1 = 1
	256 words	len-1 = 255

DATA: The data fields contain the parameters of the message. It is important that this data block contains at least one word. The low byte of the word is transmitted first.

data[i]: Parameter word of the command. The low byte is transmitted first.

Warning: The data block must contain at least one word!

CRC: The 16-bit CRC checksum. The algorithm used is CRC-CCITT. The CRC calculation includes all bytes of the frame. The data bytes have to be calculated as a word. At first you have to shift in the high byte of the data word. This is the opposite way you transmit the data word. The 16-bit generator polynomial '$x^{16}+x^{12}+x^5+1$' is used for the calculation.

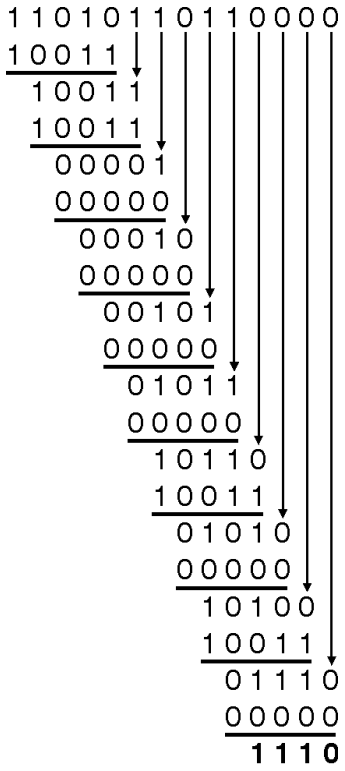
Order of CRC calculation:,
'OpCode', 'len-1', 'data[0]' high byte, 'data[0]' low byte, ... ,
zeroWord low byte = 0x00, zeroWord high byte = 0x00

CRC: Checksum of the frame. The low byte is transmitted first.

CRC-CCITT Example:

Packet M(x): 1101011011 (= $x^9+x^8+x^6+x^4+x^3+x^1+x^0$)
Generator Polynom G(x): 10011 (= $x^4+x^1+x^0$)
Polynom x^rM(x): 11010110110000

Division x^rM(x) / G(x):



Checksum CRC: 1110

EPOS CRC-CCITT Calculation:

Packet M(x): WORD DataArray[n]
Generator Polynom G(x): 10001000000100001 (= $x^{16}+x^{12}+x^5+x^0$)

DataArray[0]: HighByte(**OpCode**) + LowByte(**len-1**);
DataArray[1]: **data[0]**
DataArray[2]: **data[1]**
…
DataArray[n-1]: **0x0000 (ZeroWord)**

```
WORD CalcFieldCRC(WORD* pDataArray, WORD numberOfWords)
{
        WORD shifter, c;
        WORD carry;
        WORD CRC = 0;

        //Calculate pDataArray Word by Word
        while(numberOfWords--)
        {
            shifter = 0x8000;                   //Initialize BitX to Bit15
            c = *pDataArray++;                  //Copy next DataWord to c
            do
            {
                    carry = CRC & 0x8000;       //Check if Bit15 of CRC is set
                    CRC <<= 1;                  //CRC = CRC * 2
                    if(c & shifter) CRC++;      //CRC = CRC + 1, if BitX is set in c
                    if(carry) CRC ^= 0x1021;    //CRC = CRC XOR G(x), if carry is true
                    shifter >>= 1;              //Set BitX to next lower Bit, shifter = shifter/2
            } while(shifter);
        }
         return CRC;
 }
```

6.2.3 Transmission Byte Order

The unit of data memory in the EPOS is a word (16-bit value). To send and receive a word (16-bit) over the serial port of the EPOS, the low byte will be transmitted first.
Multiple byte data (word = 2 bytes, long words = 4 bytes) are transmitted starting with the less significant byte (LSB) first.

A word will be transmitted in this order: byte0 (LSB), byte1 (MSB).
A long word will be transmitted in this order: byte0 (LSB), byte1, byte2, byte3 (MSB).

6.2.4 Command Instruction Example

We give here an example of a command frame for the serial RS232 communication to show the composition and structure of EPOS messages during transmission and reception.

The command sent to the EPOS is ReadObject. The command can be used to read an object with 4 Bytes and less:

ReadObject 'SoftwareVersion' (Index = 0x2003, SubIndex = 0x01)

OpCode	len-1	data[0]	data[1]	CRC
0x10	0x01	0x2003	0x0201	0xA888

OpCode:	0x10	= *ReadObject*
Len-1:	0x01	= 2 Words
data[0]:	0x2003	= Index
LowByte data[1]:	0x01	= SubIndex
HighByte data[1]:	0x02	= Node-ID

The EPOS answers to the command ReadObject with an answer frame and the returned parameters in the data block as follows:

Answer to ReadObject 'SoftwareVersion' (Index = 0x2003, SubIndex = 0x01)

OpCode	len-1	data[0]	data[1]	data[2]	data[3]	CRC
0x00	0x03	0x0000	0x0000	0x2010	0x6210	0x2610

OpCode:	0x00	= Answer
Len-1:	0x03	= 4 Words
data[0]:	0x0000	= LowWord ErrorCode
data[1]:	0x0000	= HighWord ErrorCode
data[2]:	0x2010	= Value of Object 'SoftwareVersion'
data[3]:	0x6210	= Word without meaning

6.2.5 Protocol and Flow Control

Sequence for sending EPOS commands
The EPOS is always communicating as a slave. A frame is only sent as an answer to a request. Some of the EPOS commands send an answer, other commands do not. Have a look at the description of the commands to know which command sends an answer packet. The master always has to start the communication sending a packet structure.
The next two sections describe the data flow of transmission and reception frames.

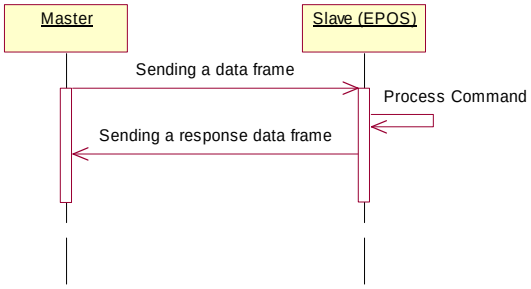


Figure 3: EPOS command with response

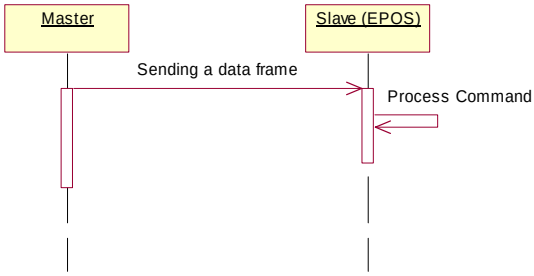


Figure 4: EPOS command without response

Timeout Handling

The timeout is handled over a complete frame. This means the timeout is evaluated over the sent data frame, over the command processing procedure and over the response data frame. For every frame (frames, data processing) the timer is reset and the timeout handling is restarted.

Object	Index	SubIndex	Default
RS232 Frame Timeout	0x2005	0x00	500 [ms]

Remark: For special requirements the timeout can be changed writing to the object dictionary!

Sending a data frame

When sending a frame you have to wait for different acknowledges. The first is a ‘Ready Acknowledge’. After sending the first byte of the frame (OpCode) you have to wait for the ready acknowledge of the EPOS. If the char ‘O’ (okay) is received, then the slave is ready to receive other data. If the char ‘F’ is received the slave is not ready to send data. The communication has to be stopped. If everything is okay you can send the rest of the data frame.

After sending the checksum you have to wait for the ‘End Acknowledge’. The slave sends either the char ‘O’ (okay) or the char ‘F’ (failed).

The following figure shows the interaction diagram of sending a packet structure.

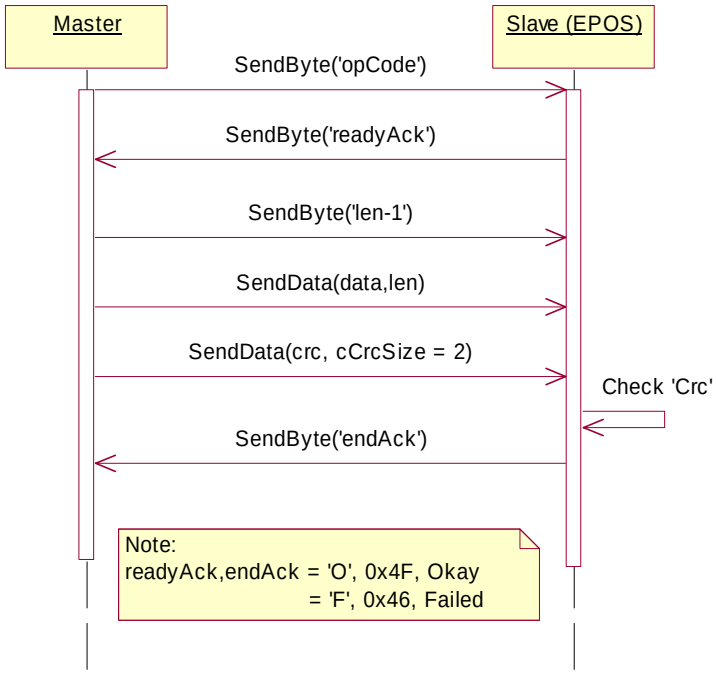


Figure 5: Sending a data frame to the EPOS

Receiving a response data frame

In response to some of the command frames, the EPOS sends a response data frame back to the master. The sequence of data flow is the same as for sending a data packet. Only the direction is changed. The master has also to send the two acknowledges to the slave.

The value of the first field must always be 0x00. This is the operation code which describes a response frame. After receiving the first byte, the master has to send the 'Ready Acknowledge'. Send the char 'O' if you are ready to receive the rest of the frame. If you are not ready to receive the rest of the frame then send 'F'. If the EPOS does not get an 'O' within the specified timeout, the communication is reset. Sending of acknowledge 'F' does not reset the communication.

After sending the ready acknowledge 'O' the rest of the data frame is sent by the EPOS. Then the checksum must be calculated and compared with the one received. If the checksum is correct send acknowledge 'O' (okay), otherwise send acknowledge 'F' to the EPOS.

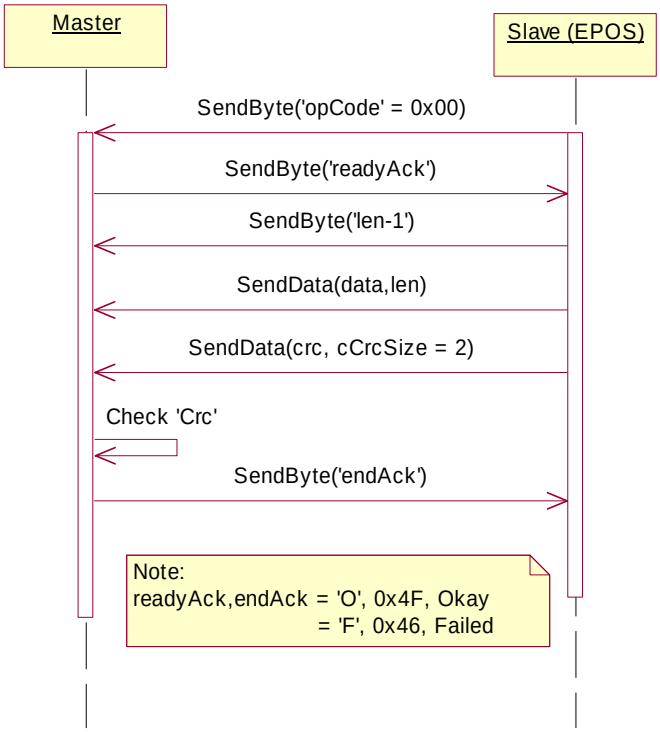


Figure 6: Receiving a response data frame from the EPOS

Master Implementation State Machine

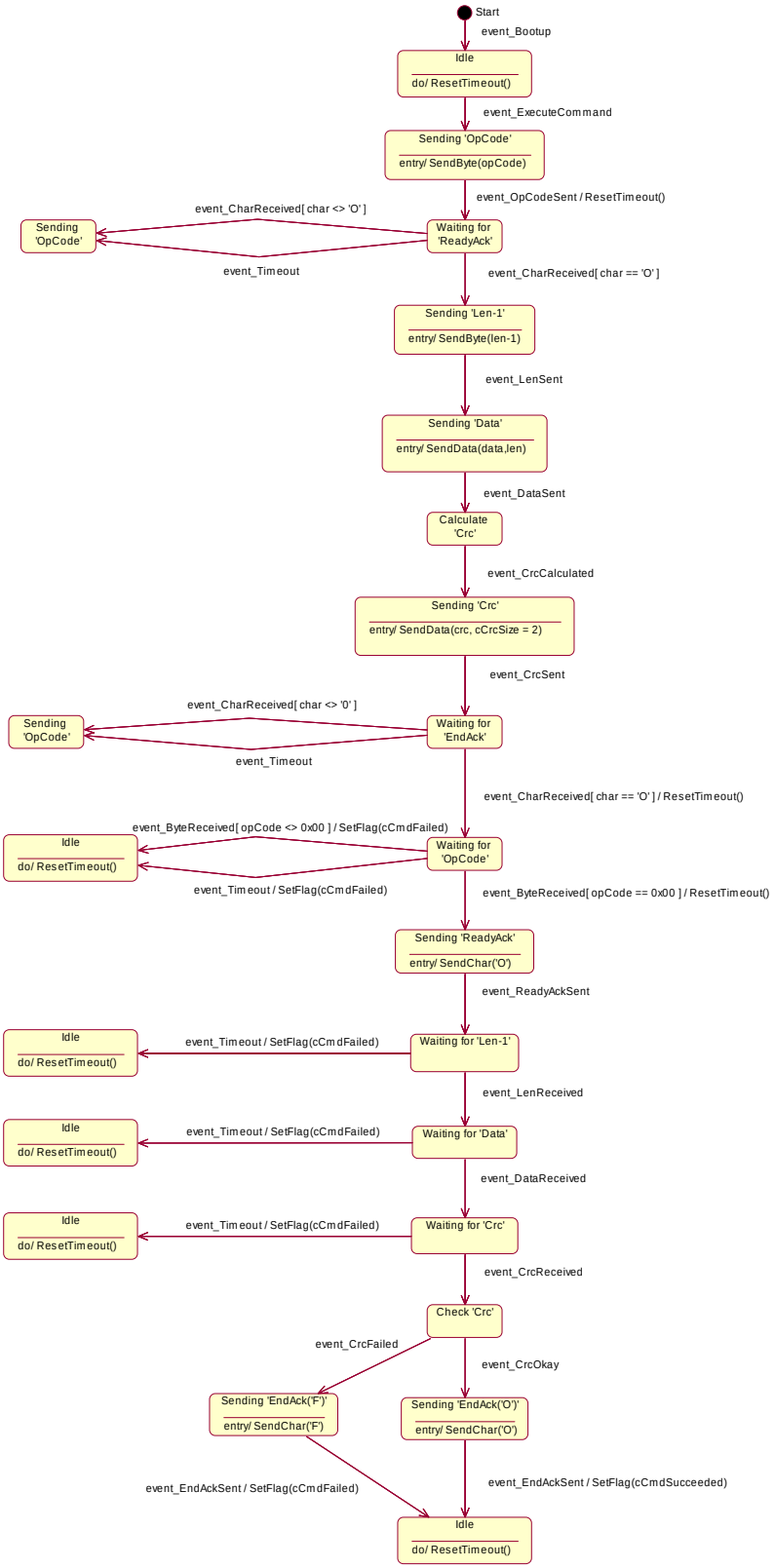


Figure 7: Master Implementation State Machine

Slave (EPOS) Implementation State Machine

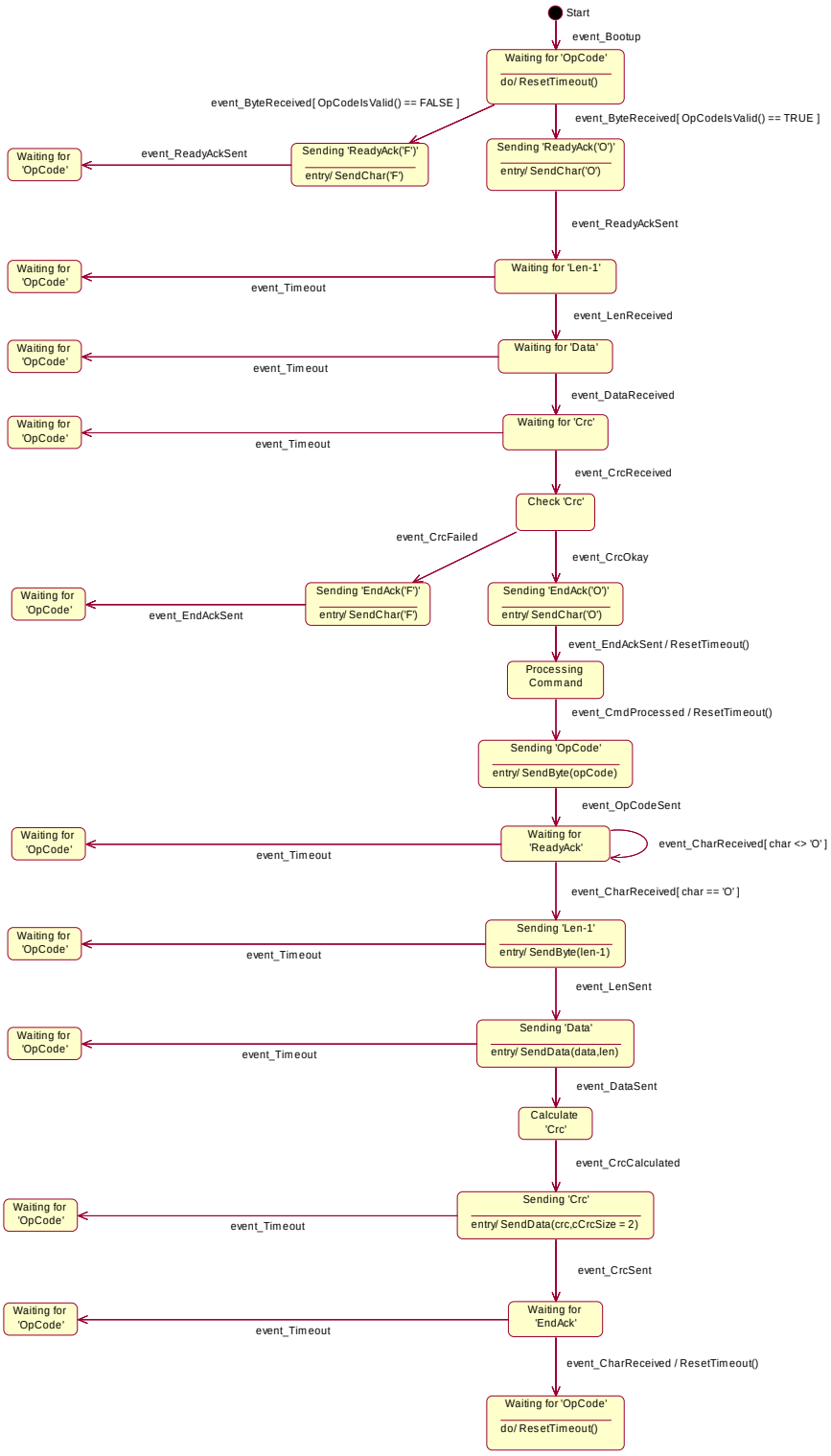


Figure 8: Master Implementation State Machine

6.3 EPOS Command Reference

6.3.1 Read Functions

6.3.1.1 Read Object Dictionary Entry (4 Data Bytes and less)

Command name	**ReadObject**	
Description	Read an object value at the given Index and SubIndex from the Object Dictionary.	
OpCode	0x10	
Len-1	1	
Parameters	WORD Index	Index of Object
	(Low) BYTE SubIndex	SubIndex of Object
	(High) BYTE NodeId	Node-ID
OpCode (Response)	0x00	
Len-1 (Response)	3	
Response	DWORD ErrorCode	see *ErrorCode Definition*
	BYTE Data[4]	Data Bytes Read

6.3.1.2 Read Object Dictionary Entry (5 Data Bytes and more)

Command name	**InitiateSegmentedRead**	
Description	Start reading an object value at the given Index and SubIndex from the Object Dictionary. Use the command 'SegmentRead' to read the data.	
OpCode	0x12	
Len-1	1	
Parameters	WORD Index	Index of Object
	(Low) BYTE SubIndex	SubIndex of Object
	(High) BYTE NodeId	Node-ID
OpCode (Response)	0x00	
Len-1 (Response)	1	
Response data	DWORD ErrorCode	see *ErrorCode Definition*

Command name	**SegmentRead**		
Description	Read a data segment of the object initiated with the command 'InitiateSegmentedRead'.		
OpCode	0x14		
Len-1	0		
Parameters	(Low) BYTE ControlByte	not used toggle not used	[Bit 0..5] [Bit 6] Toggle Bit [Bit 7]
	(High) BYTE Dummy	Byte without meaning	
OpCode (Response)	0x00		
Len-1 (Response)	X (depends on Datafield)		
Response data	DWORD ErrorCode	see *ErrorCode Definition*	
	(Low) BYTE ControlByte	length toggle more	[Bit 0..5] Number of Data Bytes [Bit 6] Toggle Bit [Bit 7] More Segments to read
	BYTE Data[0 ... 63]	Data Bytes read	

6.3.2 Write Functions

6.3.2.1 Write Object Dictionary Entry (4 Data Bytes and less)

Command name	**WriteObject**	
Description	Write an object value to the given Index and SubIndex from the Object Dictionary.	
OpCode	0x11	
Len-1	3	
Parameters	WORD Index	Index of Object
	(Low) BYTE SubIndex	SubIndex of Object
	(High) BYTE NodeId	Node-ID
	BYTE Data[4]	Data Bytes to write
OpCode (Response)	0x00	
Len-1 (Response)	1	
Response data	DWORD ErrorCode	see *ErrorCode Definition*

6.3.2.2 Write Object Dictionary Entry (5 Data Bytes and more)

Command name	**InitiateSegmentedWrite**	
Description	Start writing an object value to the given Index and SubIndex in the Object Dictionary. Use the command 'SegmentWrite' to write the data.	
OpCode	0x13	
Len-1	3	
Parameters	WORD Index	Index of Object
	(Low) BYTE SubIndex	SubIndex of Object
	(High) BYTE NodeId	Node-ID
	DWORD ObjectLength	Number of Bytes to write totally
OpCode (Response)	0x00	
Len-1 (Response)	1	
Response data	DWORD ErrorCode	see *ErrorCode Definition*

Command name	**SegmentWrite**	
Description	Write a data segment to the object initiated with the command 'InitiateSegmentedWrite'	
OpCode	0x15	
Len-1	0 ... 31	
Parameters	(Low) BYTE ControlByte	length [Bit 0..5] Number of Data Bytes toggle [Bit 6] Toggle Bit not used [Bit 7]
	BYTE Data[0 ... 63]	Data Bytes to write
OpCode (Response)	0x00	
Len-1 (Response)	2	
Response data	DWORD ErrorCode	see *ErrorCode Definition*
	(Low) BYTE ControlByte	length [Bit 0..5] Number of Bytes written toggle [Bit 6] Toggle Bit not used [Bit 7]
	(High) BYTE Dummy	Byte without meaning

6.3.2.3 SendNMTService

Command name	SendNMTService	
Description	Send a NMT service to change NMT state or reset device etc.	
OpCode	0x0E	
Len-1	1	
Parameters	WORD NodeId	Node-ID
	WORD CmdSpecifier	Command Specifier 1 = Start Remote Node 2 = Stop Remote Node 128 = Enter Pre-Operational 129 = Reset Node 130 = Reset Communication
Response data	No Response	

6.3.3 General CAN Commands

6.3.3.1 SendCANFrame

Command name	SendCANFrame	
Description	Send a general CAN Frame to the CAN bus.	
OpCode	0x20	
Len-1	9	
Parameters	WORD Identifier	CAN Frame 11-bit Identifier
	WORD Length	CAN Frame Data Length Code (DLC)
	BYTE Data [8]	CAN Frame Data
Response data	No Response	

6.3.3.2 RequestCANFrame

Command name	RequestCANFrame	
Description	Request a general CAN Frame from the CAN bus using Remote Transmit Request (RTR).	
OpCode	0x21	
Len-1	1	
Parameters	WORD Identifier	CAN Frame 11-bit Identifier
	WORD Length	CAN Frame Data Length Code (DLC)
OpCode (Response)	0x00	
Len-1 (Response)	5	
Response data	DWORD ErrorCode	see *ErrorCode Definition*
	BYTE Data [8]	CAN Frame Data

6.4 ErrorCode Definition

The following error codes are defined by CANopen Communication Profile 301:

0x00000000:	No error
0x06020000:	Object does not exist in the object dictionary
0x06090011:	Sub-index does not exist
0x05040001:	Client/server command specifier not valid or unknown
0x05030000:	Toggle bit not alternated
0x05040000:	SDO protocol timed out
0x05040005:	Out of memory
0x06010000:	Unsupported access to an object
0x06010001:	Attempt to read a write only object
0x06010002:	Attempt to write a read only object
0x06040041:	Object cannot be mapped to the PDO
0x06040042:	The number and length of the objects to be mapped would exceed PDO length
0x06040043:	General parameter incompatibility reason
0x06040047:	General internal incompatibility in the device
0x06060000:	Access failed due to an hardware error
0x06070010:	Data type does not match, length of service parameter does not match
0x06070012:	Data type does not match, length of service parameter too high
0x06070013:	Data type does not match, length of service parameter too low
0x06090030:	Value range of parameter exceeded (only for write access)
0x06090031:	Value of parameter written too high
0x06090032:	Value of parameter written too low
0x06090036:	Maximum value is less than minimum value
0x08000000:	General error
0x08000020:	Data cannot be transferred or stored to the application
0x08000021:	Data cannot be transferred or stored to the application because of local control
0x08000022:	Data cannot be transferred or stored to the application because of the present device state

The following error codes are maxon specific:

0x0F00FFC0:	The device is in wrong NMT state
0x0F00FFBF:	The RS232 command is illegal
0x0F00FFBE:	The password is not correct
0x0F00FFBC:	The device is not in service mode
0x0F00FFB9:	Error Node-ID

7 CAN Communication

7.1 Introduction

The CAN interface of the maxon EPOS drives follows the CiA CANopen specifications DS-301 communication profile and DSP-402 device profile (Device Profile for Drives and Motion Control). This document supplements the specifications DS-301 Version 4.02 and DSP-402 Version 2.0, which must be purchased separately from CiA (international users and manufacturers group).

CiA (Can in Automation e.V.) can be contacted on http://www.can-cia.org.

7.2 CAN Documentations

[1] Bosch CAN Specification 2.0B
[2] CiA CANopen DS-301 Communication Profile for Industrial Systems
[3] CiA CANopen DSP-402 Device Profile for Drives and Motion Control
[4] Konrad Etschberger: Controller Area Network (ISBN3-446-21776-2)

7.3 Abbreviations and Terms

Numbers followed by “h” are hexadecimal.
Numbers followed by “b” are binary.
All other numbers are decimal.

The following **abbreviations** are used in this manual:

CAL	CAN application layer
CMS	CAN message specification.
COB	Communication object (CAN Message); A unit of transportation in a CAN message network; Data must be sent across a network inside a COB.
COB-ID	COB-Identifier; Identifies a COB uniquely in a network; The identifier determines the priority of that COB in the MAC sub-layer too.
EDS	Electronic data sheet; A standard form of all CAN objects supported by a device. The EDS is used by external CAN configurators.
ID	Identifier; The name by which a CAN device is addressed.
MAC	Medium Access Control; One of the sub-layers of the Data Link Layer in the CAN Reference Model that controls who gets access to the medium to send a message.
OD	Object dictionary, which is the full set of objects supported by the node. It is the interface between the application and communication (see “Object” below).
PDO	Process Data Object; Object for data exchange between several devices.
SDO	Service Data Object; Peer to peer communication with access to the object directory of the device.
PLC	Programmable controller; A PLC can serve as a CAN master for the EPOS.
RO, RW, WO	Read Only, Read Write, Write Only.

The following **terms** are used in this manual:

CAN client or CAN master	A host, typically a PC or other control equipment, that supervises the nodes of a network.
CAN server or CAN slave	A node in the CAN network that can give service under control of the CAN master.
Object	A CAN message with a meaningful functionality and/or data. Objects are referenced according to addresses in the object dictionary.
Receive	“received” data is sent from the control equipment to the EPOS.
Transmit	“transmitted” data is sent from the EPOS to the other equipment.

7.4 CANopen Basics

This chapter describes, in general, the CANopen communication features most relevant to the maxon motor EPOS positioning controllers. More detailed information is available in the specific CANopen documentation.

The CANopen communication concept can be described similar to the ISO Open Systems Interconnection (OSI) Reference Model. CANopen represents a standardized application layer and communication profile.

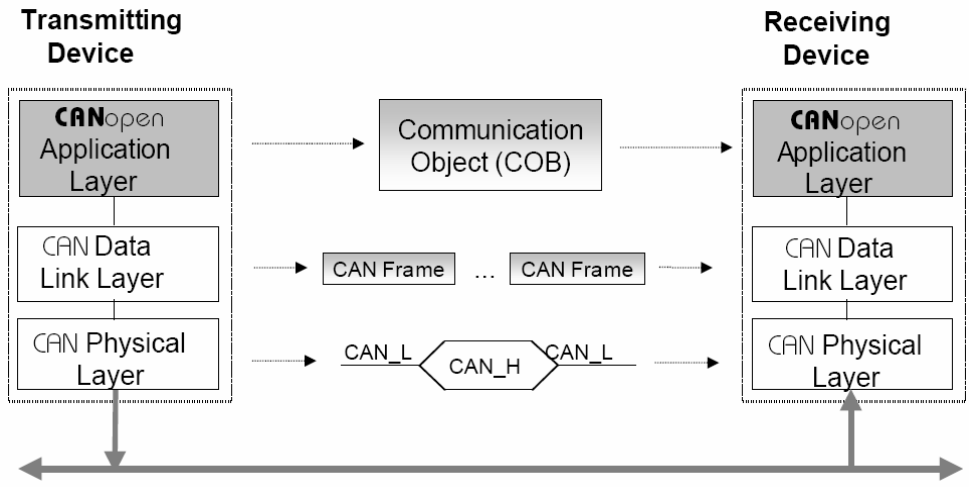


Figure 9: Protocol layer interactions

7.4.1 Physical Layer

CANopen is a networking system based on the CAN serial bus. CANopen assumes that the device's hardware has a CAN transceiver and CAN controller as specified in ISO 11898. The physical medium is a differently driven two-wire bus line with common return.

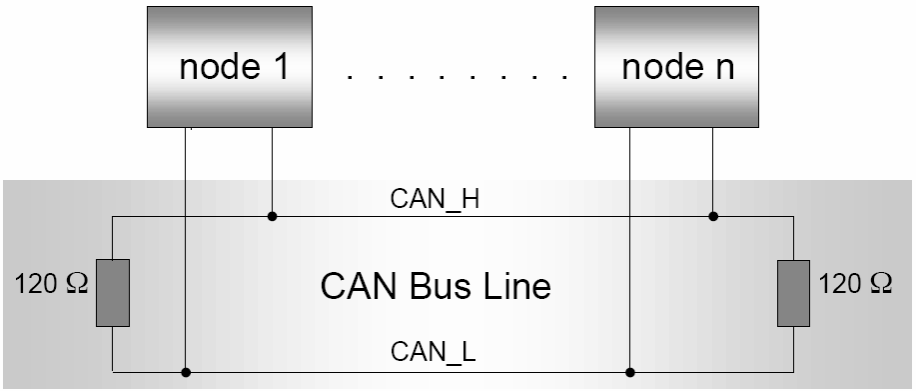


Figure 10: ISO 11898 Basic Network Setup

7.4.2 Data Link Layer

The CAN data link layer is also standardized in ISO 11898. The data link layer services are implemented in the Logical Link Control (LLC) and Medium Access Control (MAC) sub-layers of a CAN controller. The LLC provides acceptance filtering, overload notification and recovery management. The MAC is responsible for data encapsulation (de-capsulation), frame coding (stuffing/de-stuffing), medium access management, error detection, error signaling, acknowledgement, and serialization (de-serialization).

A Data Frame is produced by a CAN node when the node wishes to transmit data or if this is requested by another node. Within one frame up to 8 byte data can be transported. The Data Frame begins with a dominant Start of Frame (SOF) bit for hard synchronization of all nodes. The SOF bit is followed by the Arbitration Field reflecting content and priority of the message. The next field is the Control Field which specifies mainly the number of bytes of data contained in the message. The Cyclic Redundancy Check (CRC) Field is used to detect possible transmission errors. It consists of a 15-bit CRC sequence completed by the recessive CRC delimiter bit. During the Acknowledgement (ACK) Field the transmitting node sends out a recessive bit. Any node that has received an error free frame acknowledges the correct reception of the frame by sending back a dominant bit. The recessive bits of the End of Frame end the Data Frame. Between two frames there must be a recessive 3-bit Intermission field.

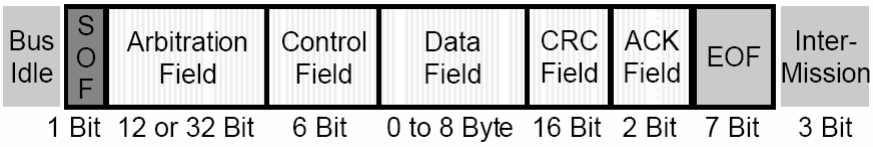


Figure 11: CAN Data Frame

On the EPOS only the Standard Frame Format is supported. The Identifier's (COB-ID) length in the Standard Format is 11 bit. The Identifier is followed by the RTR bit (Remote Transmission Request bit). In Data Frames the RTR bit has to be dominant. Within a Remote Frame the RTR bit has to be recessive. The Base ID is followed by the IDE (Identifier Extension) bit transmitted dominant in the Standard Format (within the Control Field).
The Control Field in Standard Format includes the Data Length Code (DLC), the IDE bit, which is transmitted dominant and the reserved bit r0 also transmitted dominant. The reserved bits have to be sent dominant, but receivers accept dominant and recessive bits in all combinations.

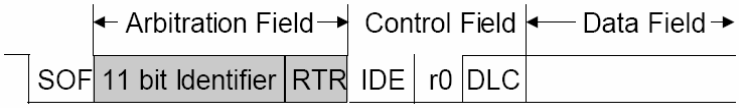


Figure 12: Standard Frame Format

7.5 CANopen Application Layer

7.5.1 Object Dictionary

The most important part of a CANopen device is the object dictionary. The object dictionary is essentially a grouping of objects accessible via the network in an ordered pre-defined fashion. Each object within the dictionary is addressed using a 16-bit index and an 8-bit sub-index. The overall layout of the standard object dictionary conforms to other industrial field-bus concepts.

Index	Variable accessed
0000h	Reserved
0001h-009Fh	Data types (not supported on EPOS)
00A0h-0FFFh	Reserved
1000h-1FFFh	Communication Profile Area (DS-301)
2000h-5FFFh	Manufacturer Specific Profile Area (maxon motor)
6000h-9FFFh	Standardized Device Area (DSP-402)
A000h-FFFFh	Reserved

Table 1: Object Directory Layout

A 16-bit index is used to address all entries within the object dictionary. In case of a simple variable this references the value of this variable directly. In case of records and arrays however, the index addresses the whole data structure. To allow individual elements of structures of data to be accessed via the network a sub-index has been defined. For single object dictionary entries such as unsigned8, boolean, integer32, etc. The value for the sub-index is always zero. For complex object dictionary entries such as arrays or records with multiple data fields the sub-index references fields within a data structure pointed to by the main index. For example on a receive PDO there exist a data-structure at index 1400h which defines the communication parameters for that module. This structure contains fields for the COB-ID and the transmission type. The sub-index concept can be used to access these individual fields as shown below.

Index	SubIndex	Variable accessed	Data type
1400h	0	Number of entries	UNSIGNED8
1400h	1	COB-ID receive PDO1	UNSIGNED32
1400h	2	Transmission type receive PDO1	UNSIGNED8

Table 2: Object Directory Entry

7.5.2 Communication Objects

CANopen communication objects are described by the services and protocols. They are classified as follows:

- The real-time data transfer is performed by means of Process Data Objects.
- With Service Data Objects, the read and write access to entries of a device object dictionary is provided.
- Special Function Objects provide application-specific network synchronization and emergency messages.
- The Network Management Objects provide services for network initialization, error control and device status control.

Process Data Objects (PDO)
Service Data Objects (SDO)
Special Function Objects: - Synchronization Objects (SYNC) - Time Stamp Objects (not used on EPOS) - Emergency Objects (EMCY)
Network Management Objects: - NMT Message - Node Guarding Object

Table 3: Communication Objects

7.5.3 Predefined Communication Objects

7.5.3.1 PDO Object

PDO communication can be described by the producer/consumer model. Process data can be transmitted from one device (producer) to one another device (consumer) or to many other devices (broadcasting). PDOs are transmitted in a non confirmed mode. The producer sends a Transmit-PDO (TxPDO) with a specific identifier, which corresponds to the identifier of the Receive-PDO (RxPDO) of one or more consumers.

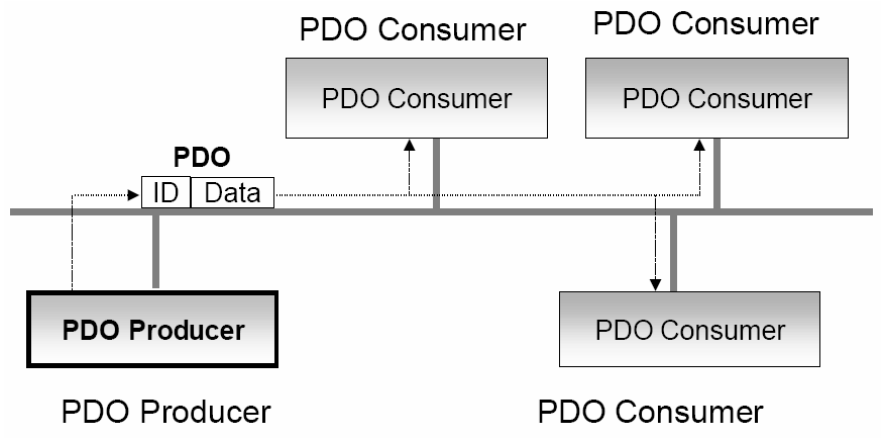


Figure 13: Process Data Object

There are two PDO services: to write a PDO and to read a PDO. The Write-PDO is mapped to a single CAN Data frame. The Read-PDO is mapped to CAN Remote Frame, which will be responded by the corresponding CAN Data Frame. Read-PDOs are optional and depending on the device capability. The complete data field of up to 8 byte may contain process data. Number and length of PDOs of a device are application-specific and have to be specified in the device profile. The number of supported PDOs is accessible at index 1004h in the Object Dictionary. The PDOs correspond to entries in the Object Dictionary and provide the interface to application objects. Data type and mapping of application objects into a PDO is determined by a corresponding default PDO mapping structure within the Object Dictionary. This structure is defined in the entries 1600h for the 1st R_PDO and 1A00h for the first T_PDO. In one CANopen network there can be used up to 512 T_PDOs and 512 R_PDOs.

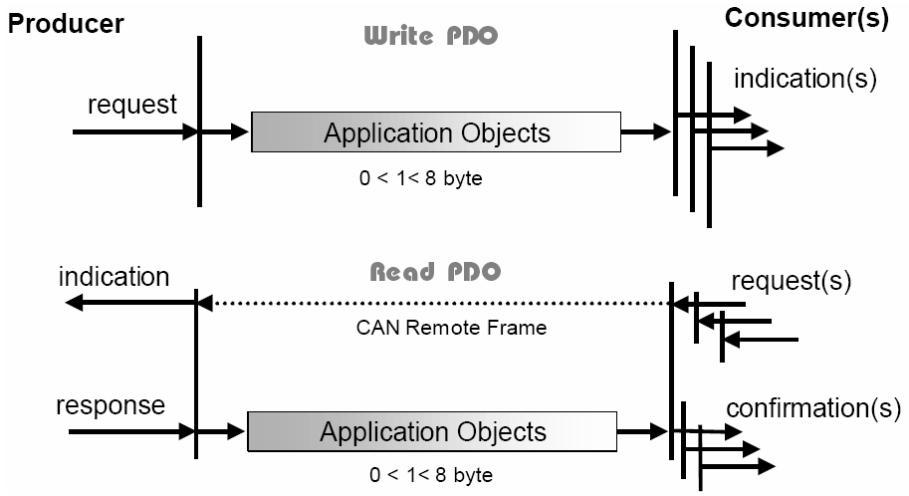


Figure 14: PDO Protocol

The CANopen communication profile distinguishes three message triggering modes:

1. Message transmission is triggered by the occurrence of an object specific event specified in the device profile.
2. The transmission of asynchronous PDOs may be initiated on receipt of a remote request initiated by another device.
3. Synchronous PDOs are triggered by the expiration of a specified transmission period synchronized by the reception of the SYNC object.

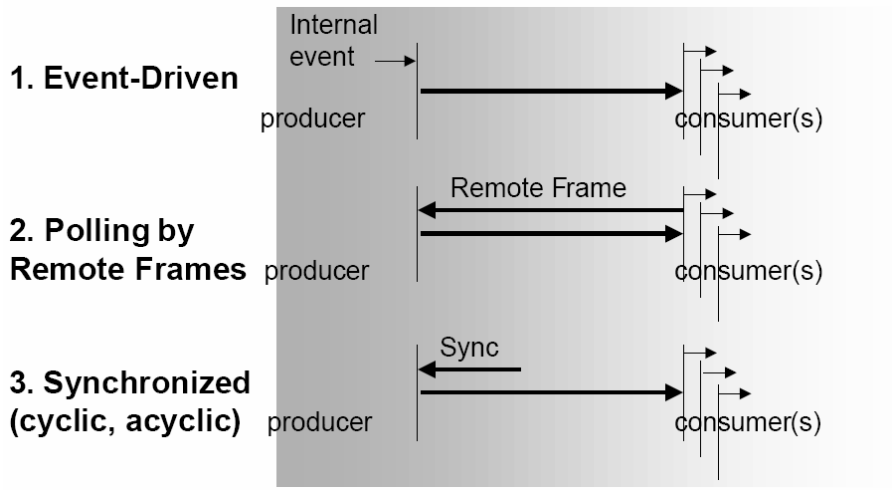


Figure 15: PDO Communication Modes

7.5.3.2 SDO Object

With Service Data Objects (SDO) the access to entries of a device object dictionary is provided. A SDO is mapped to two CAN Data Frames with different identifiers, because the communication is confirmed. By means of a SDO a peer-to-peer communication channel between two devices may be established. The owner of the accessed object dictionary is the server of the SDO. A device may support more than one SDO. One supported SDO is mandatory and the default case.

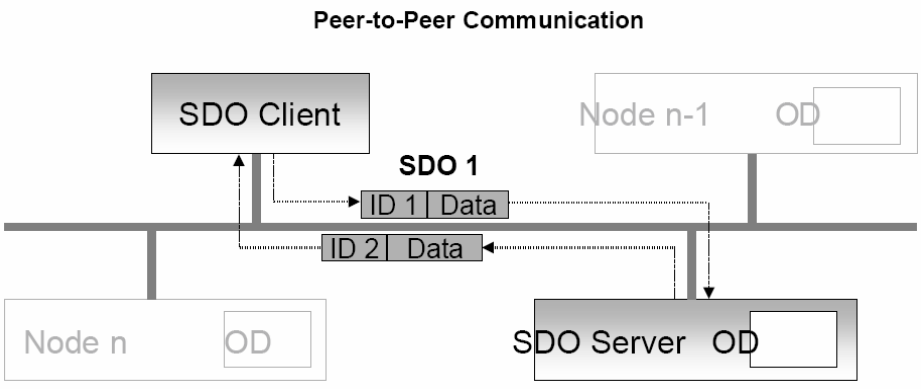


Figure 16: Service Data Object

Read and write access to the CANopen object dictionary is performed by SDOs. The Client/Server Command Specifier contains the following information:

- download / upload
- request / response
- segmented / expedited transfer
- number of data bytes
- end indicator
- alternating toggle bit for each subsequent segment

SDOs are described by the communication parameter. The default Server-SDO (S_SDO) is defined in the entry 1200h. In one CANopen network there may be used up to 256 SDO channels requiring two CAN identifiers each.

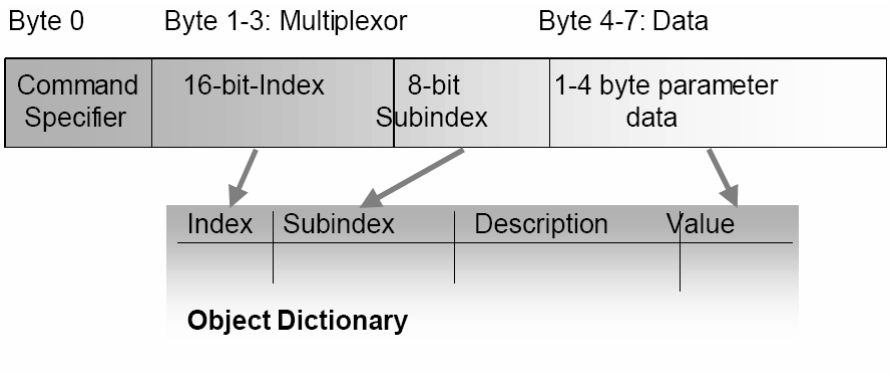


Figure 17: Object Dictionary Access

7.5.3.3 SYNC Object

The Sync-Producer provides the synchronization-signal for the Sync-Consumer. When the Sync-Consumer receives the signal they start carrying out their synchronous tasks. In general the fixing of the transmission time of synchronous PDO messages coupled with the periodicity of transmission of the SYNC Object guarantees that sensor devices may arrange to sample process variables and that actuator devices may apply their actuation in a coordinated fashion. The identifier of the SYNC Object is available at index 1005h.

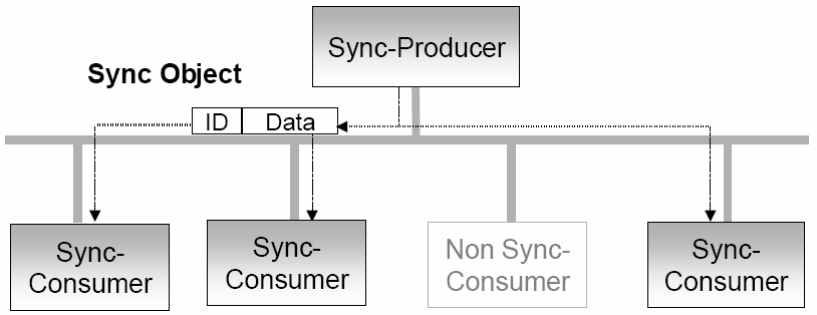


Figure 18: Synchronization Object

Synchronous transmission of a PDO means that the transmission is fixed in time with respect to the transmission of the SYNC Object. The synchronous PDO is transmitted within a given time window ‘synchronous window length’ with respect to the SYNC transmission, and at most once for every period of the SYNC. The time period between the SYNC objects is specified by the parameter ‘communication cycle period’.

CANopen distinguishes the following transmission modes:

- synchronous transmission,
- asynchronous transmission

Synchronous PDOs are transmitted within the synchronous window after the SYNC object. The priority of synchronous PDOs is higher than the priority of asynchronous PDOs. Asynchronous PDOs and SDOs can be transmitted at every time with respect to their priority. So they could also be transmitted within the synchronous window.

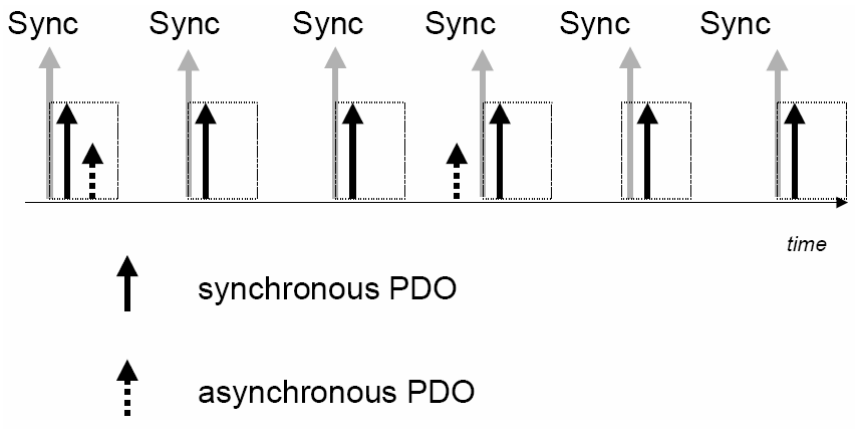


Figure 19: Synchronous PDO

7.5.3.4 EMERGENCY Object

Emergency messages are triggered by the occurrence of a device internal fatal error situation and are transmitted from the concerned application device to the other devices with high priority. This makes them suitable for interrupt type error alerts. An Emergency Telegram may be sent only once per 'error event', i.e. the emergency messages must not be repeated. As long as no new errors occur on a device no further emergency message must be sent. By means of CANopen Communication Profile defined emergency error codes, the error register and device specific additional information are specified in the device profiles.

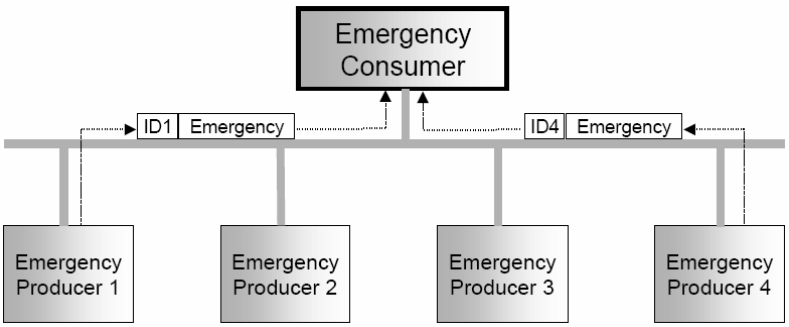


Figure 20: Emergency Service

7.5.3.5 NMT Services

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 Master. The other nodes are NMT Slaves. The network management provides the following functionality groups: Module Control Services for initialization of NMT Slaves that want to take part in the distributed application, Error Control Services for supervision of the nodes and networks communication status, Configuration Control Services for up- and downloading of configuration data from respectively to a module of the network. A NMT Slave represents that part of a node, which is responsible for the node's NMT functionality. A NMT Slave is uniquely identified by its module ID.

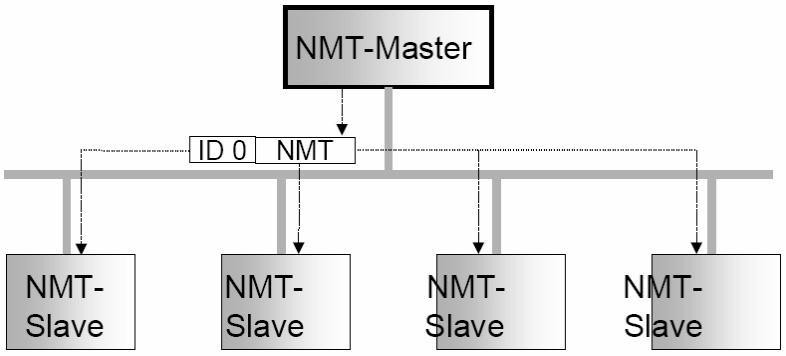


Figure 21: Network Management (NMT)

The CANopen NMT Slave devices implement a state machine, which brings automatically after power-on and internal initialization every device in Pre-operational state. In this state the node may be configured and parameterized via SDO (e.g. using a configuration tool), no PDO communication is allowed. The NMT Master device may switch all nodes or a single node to Operational state and vice versa. In Operational state PDO transfer is allowed. By switching a device into the Stopped state it is forced to stop PDO and SDO communication Furthermore, this state can be used to achieve certain application behaviour. The definition of this behaviour falls into the scope of device profiles. In the Operational state all communication objects are active. Object Dictionary access via SDO is possible. Implementation aspects or the application state machine however may require to switch off or to read only certain application objects whilst being operational (e.g. an object may contain the application program, which cannot be changed during execution).

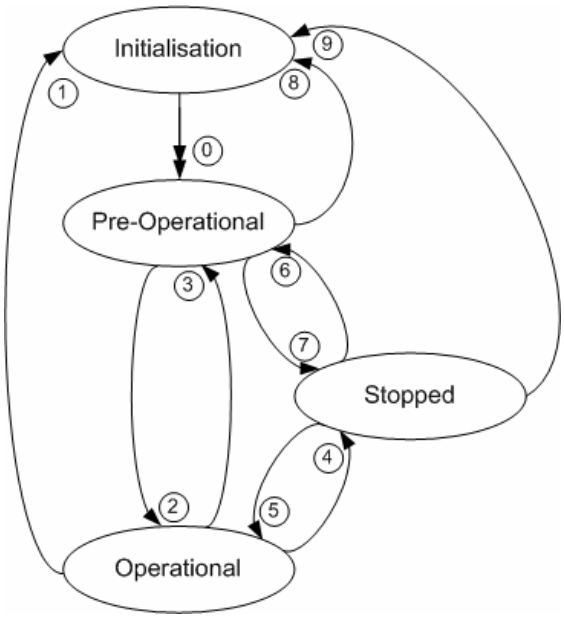


Figure 22: NMT Slave State Diagram

CANopen Network Management provides the following five services, which can be distinguished by the Command Specifier (CS):

Service[1]	Transitions in State Diagram	NMT State after Command	Statusword Remote Bit 9	Functionality
- [2]	0	Pre-Operational	FALSE	Communication: - Service Data Objects (SDO) Protocol - Emergency Objects - Network Management (NMT) Protocol
Enter Pre-Operational	3, 6	Pre-Operational	FALSE	
Reset Communication	1, 8, 9	Initialisation (change automatic to Pre-Operational)	FALSE	- Calculate the SDO Cob-Id's - Setup Dynamic PDO-Mapping and calculate the PDO Cob-Id's - Communication: While initialization is active no communication is supported. After complete a Boot-Up message is send to the CAN bus
Reset Node	1, 8, 9	Initialisation (change automatic to Pre-Operational)	FALSE	This command generates a general reset of EPOS software. It's the same effect like turn off and on the supply voltage. All not saved parameters are gone and overwritten with values saved to the EEPROM with Save all Parameters.
Start Remote Node	2, 5	Operational	TRUE	Communication: - Service Data Objects (SDO) Protocol - Process Data Objects (PDO) Protocol - Emergency Objects - Network Management (NMT) Protocol
Stop Remote Node	4, 7	Stopped	FALSE	Communication: - Network Management (NMT) Protocol

Notes:
[1] Command may be sent with Network Management (NMT) Protocol.
[2] This Transition is generated automatically by the EPOS device after initialisation is completed. After initialisation a Boot-Up message is send.

The communication object has got the identifier=0 and consists out of two bytes. The Node-ID defines the destination of the message. If it is zero the protocol addresses all NMT Slaves.

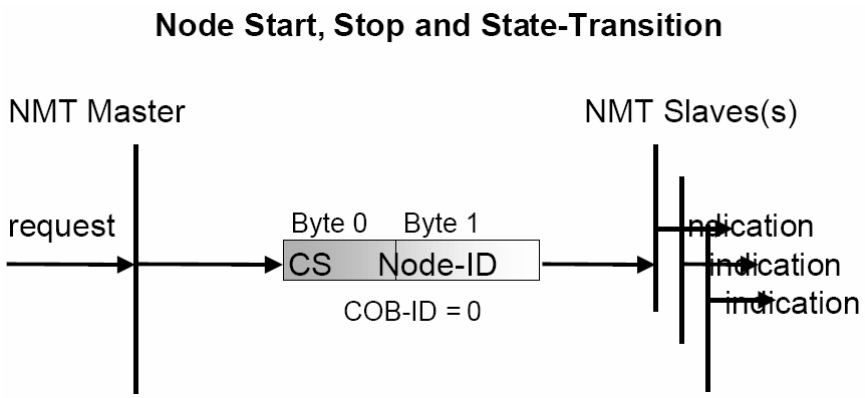


Figure 23: NMT Object

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 Pre-Operational NMT State
0	0x80	n	The EPOS (or CANopen Node) with the Node-ID n will Enter the Pre-Operational NMT State

Reset Communication Protocol

COB-ID	**CS** (Byte 0)	**Node-ID** (Byte 1)	**Functionality**
0	0x82	0 (all)	All EPOS (all CANopen Nodes) will Reset the Communication
0	0x82	n	The EPOS (or CANopen Node) with the Node-ID n will Reset the Communication

Reset Node Protocol

COB-ID	**CS** (Byte 0)	**Node-ID** (Byte 1)	**Functionality**
0	0x81	0 (all)	All EPOS (all CANopen Nodes) will Reset
0	0x81	n	The EPOS (or CANopen Node) with the Node-ID n will Reset

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

Stop Remote Node Protocol

COB-ID	**CS** (Byte 0)	**Node-ID** (Byte 1)	**Functionality**
0	0x02	0 (all)	All EPOS (all CANopen Nodes) will Enter the Stopped NMT State
0	0x02	n	The EPOS (or CANopen Node) with the Node-ID n will Enter the Stopped NMT State

7.5.3.6 Node Guarding Protocol

To detect absent devices (e.g. because of bus-off) that 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 in the Stopped Mode active.

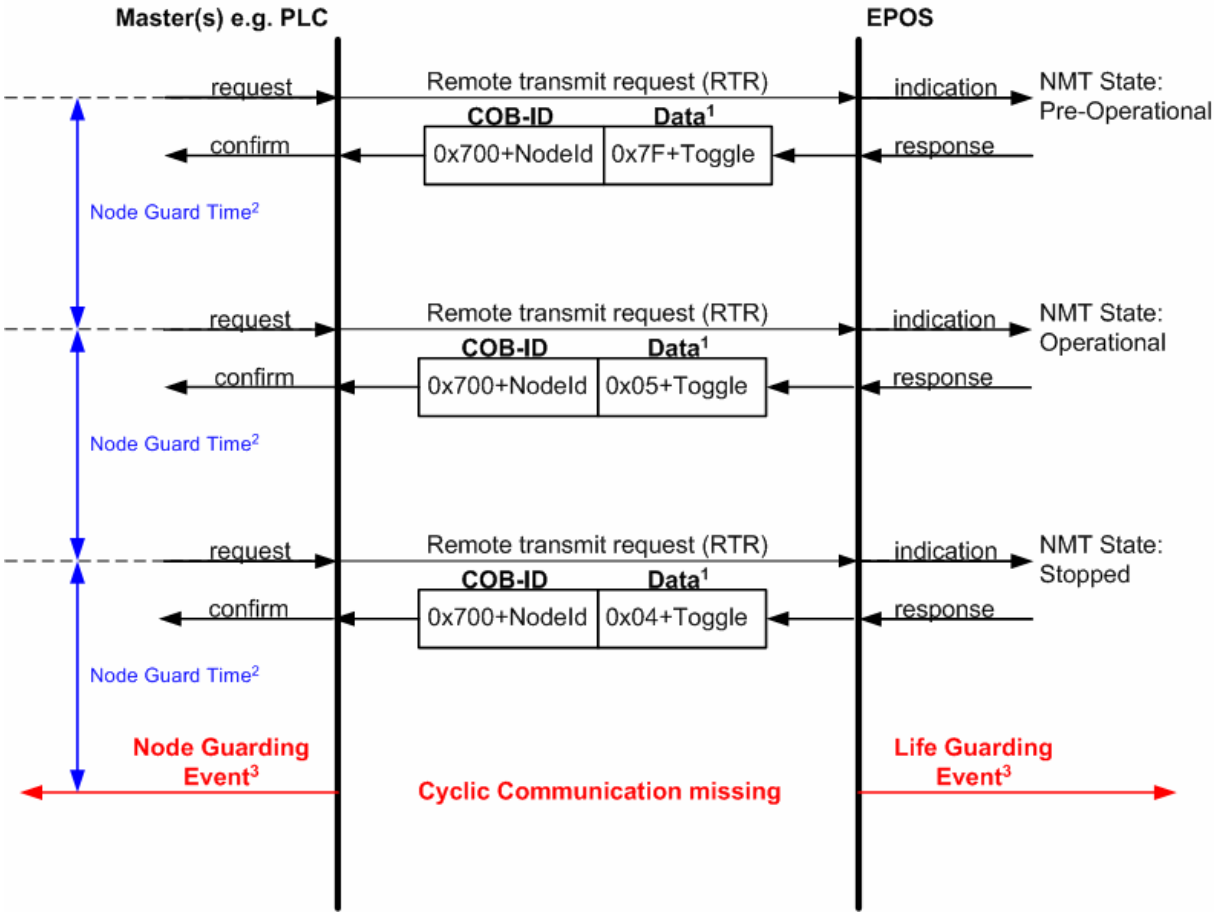


Figure 24: Node Guarding Protocol Timing Diagram

Notes:

[1] **Data Field**

The Data Field holds the NMT State. Each time the value of Toggle change 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] **Data Field**

The 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 his 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.

7.5.3.7 Heartbeat Protocol

The Heartbeat Protocol holds 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.

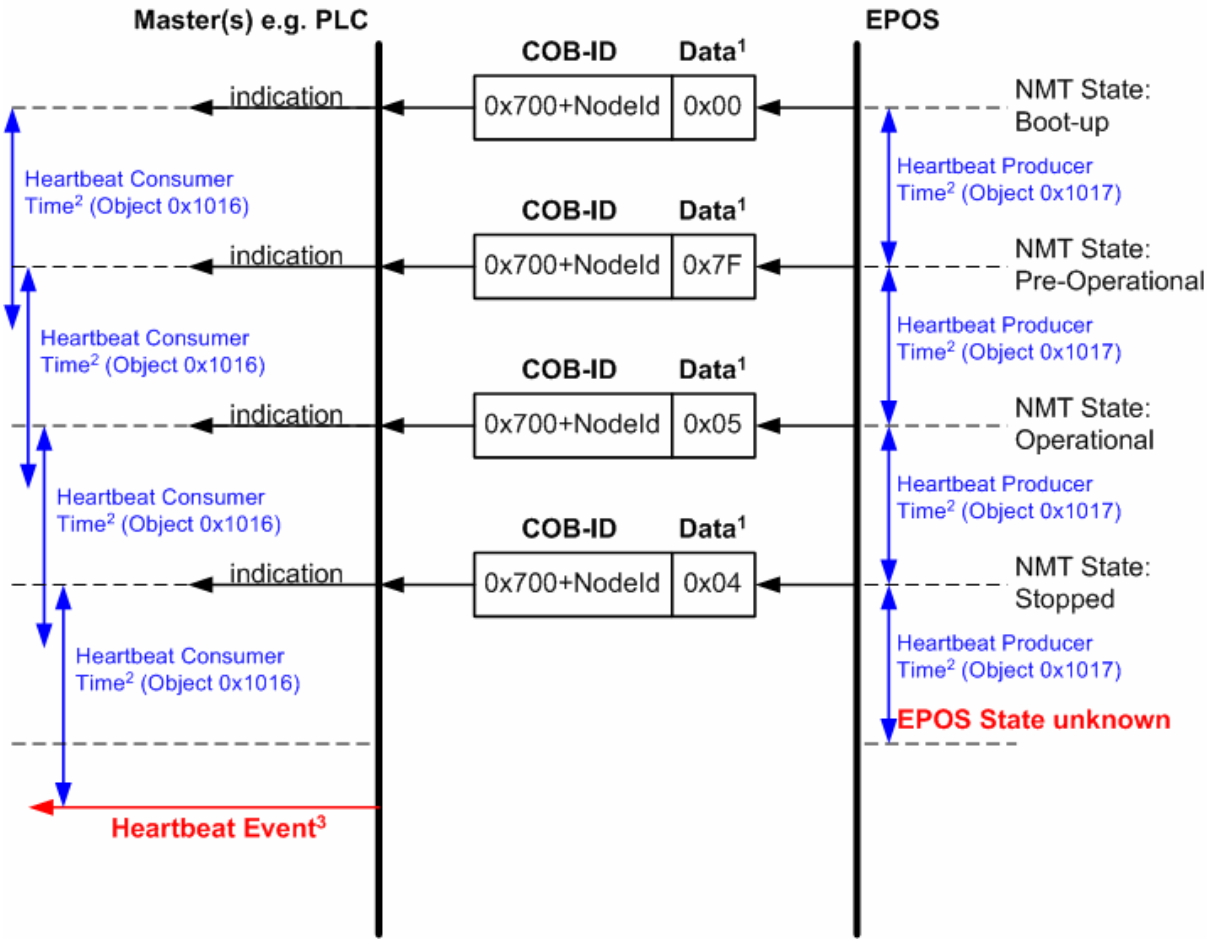


Figure 25: Heartbeat Protocol Timing Diagram

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 (a value more than 5ms would be ok). 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.

7.6 Identifier allocation scheme

The default ID-allocation scheme consists of a functional part (Function Code) and a Module-ID part, which allows distinguishing between devices. The Module-ID is assigned by DIP-Switches and a SDO Object.

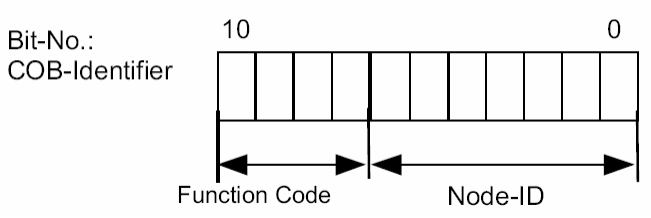


Figure 26: Default Identifier allocation scheme

This ID allocation scheme allows a peer-to-peer communication between a single master device and up to 127 slave devices. It also supports the broadcasting of non-confirmed NMT Services, SYNC and node guarding. The pre-defined master/slave connection set supports one emergency object, one SDO, and 3 Receive-PDOs and 3 Transmit-PDOs and the node guarding object.

Object	Function code (binary)	Resulting COB-ID		Communication Parameter at Index
NMT	0000	0		-
SYNC	0001	128	(0080h)	1005h
EMERGENCY	0001	129 – 255	(0081h-00FFh)	1014h
PDO1 (tx)	0011	385 – 511	(0181h-01FFh)	1800h
PDO1 (rx)	0100	513 – 639	(0201h-027Fh)	1400h
PDO2 (tx)	0101	641 – 767	(0281h-02FFh)	1801h
PDO2 (rx)	0110	769 – 895	(0301h-037Fh)	1401h
PDO3 (tx)	0111	897 – 1023	(0381h-03FFh)	1802h
PDO3 (rx)	1000	1025 – 1151	(0401h-047Fh)	1402h
PDO4 (tx)	1001	1153 – 1279	(0481h-04FFh)	1803h
PDO4 (rx)	1010	1281 – 1407	(0501h-057Fh)	1403h
SDO1 (tx)	1011	1409 – 1535	(0581h-05FFh)	1200h
SDO1 (rx)	1100	1537 – 1663	(0601h-067Fh)	1200h

Table 4: Objects of the Default Connection Set

8 Gateway Communication RS232 to CAN

Using the gateway functionality, the master can access all other EPOS devices connected to the CAN Bus via the RS232 port of the gateway device. Even other CANopen devices (i.e. I/O modules) supporting the CANopen standard CiA DS 301 can be accessed. The figure below shows the communication structure.

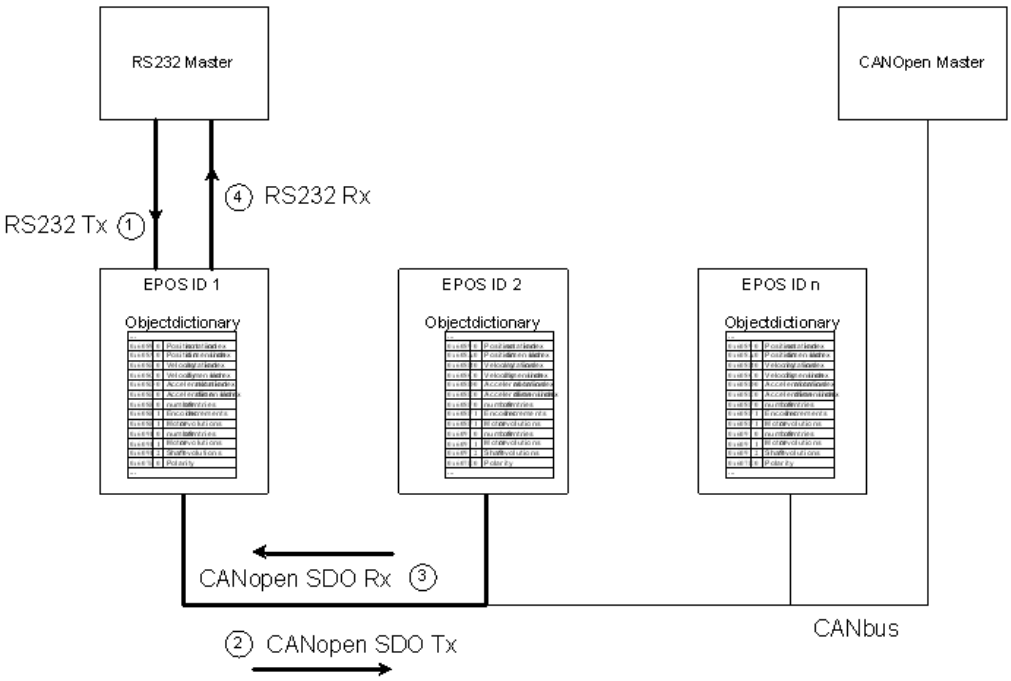


Figure 27: Communication Structure Gateway

	Protocol	Sender ⇨ Receiver	Description
Step 1	RS232, maxon specific	RS232 Master ↓ EPOS ID 1, Gateway	Command including the Node-ID is sent to the device working as a gateway. The gateway decides whether to execute the command or to translate and forward it to the CAN bus. **Criteria:** Node-ID = 0 (Gateway) → Execute Node-ID = DIP-Switch → Execute else → Forward to CAN
Step 2	CANopen, SDO	EPOS ID 1, Gateway ↓ EPOS ID 2	The gateway is forwarding the command to the CAN network. The RS232 command is translated to a CANopen SDO service.
Step 3	CANopen, SDO	EPOS ID 2 ↓ EPOS ID 1, Gateway	The EPOS ID 2 is executing the command and sending the corresponding CAN frame back to the gateway.
Step 4	RS232, maxon specific	EPOS ID 1, Gateway ↓ RS232 Master	The gateway is receiving the CAN frame corresponding to the SDO service. This CAN frame is translated back to the RS232 frame and sent back to the RS232 master.

The communication data is exchanged between the RS232 master and the gateway using the RS232 protocol. This protocol is maxon specific. Between the gateway and the addressed device the data is exchanged using the CANopen SDO protocol according to the CiA Standard DS 301.

Note: For details of CAN bus wiring see ‘Application Note CANopen Basic Information’!