EPOS2

Positioning Controller

Application Note
"USB or RS232 to CAN Gateway"

March 2009 Edition

EPOS2 Module 36/2, EPOS2 24/5, EPOS2 50/5
Firmware Version 2110 or higher

Introduction

The EPOS2 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 250 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 simple point-to-point communication, the EPOS2 also supports an USB or RS232 interface. In order to access a network using the USB or RS232 protocol, the EPOS2 includes *USB-to-CANopen gateway* or *RS232-to-CANopen gateway* functionality.

Objectives

This application note explains the functionality of the built-in communication gateway USB to CANopen or RS232 to CANopen. Advantages and disadvantages of this communication structures are discussed.

Reference and required tool

The latest editions of maxon motor documents and tools are free of charge available under http://www.maxonmotor.com category «Service & Downloads» or in the maxon motor e-shop http://shop.maxonmotor.com.

Document	Suitable order number for EPOS2 Positioning Controller
EPOS2 Communication Guide EPOS2 Firmware Specification	360665, 367676, 347717

Tool	
EPOS Studio Version 1.41 or higher	360665, 367676, 347717

Communication Structure

Using the gateway functionality, the master can access all other EPOS2 devices connected to the CAN Bus via the USB port or 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.

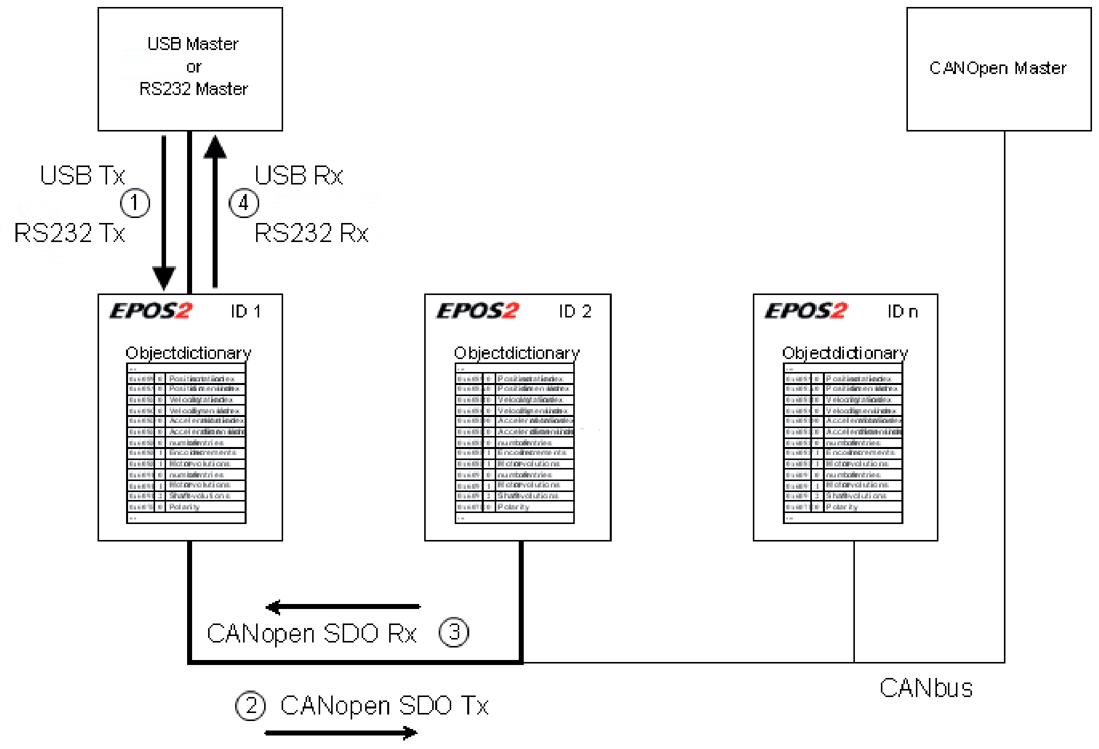

Figure 1: *Communication Structure Gateway*

	Protocol	Sender ⇨ Receiver	Description
Step 1	USB [maxon-specific] or RS232 [maxon specific]	USB or RS232 Master 🡇 EPOS2 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]	EPOS2 ID 1, Gateway 🡇 EPOS2 ID 2	The gateway is forwarding the command to the CAN network. The USB/RS232 command is translated to a CANopen SDO service.
Step 3	CANopen [SDO]	EPOS2 ID 2 🡇 EPOS2 ID 1, Gateway	The EPOS2 ID 2 is executing the command and sending the corresponding CAN frame back to the gateway.
Step 4	USB [maxon specific] or RS232 [maxon specific]	EPOS2 ID 1, Gateway 🡇 USB or RS232 Master	The gateway is receiving the CAN frame corresponding to the SDO service. This CAN frame is translated back to the USB/RS232 frame and sent back to the USB/RS232 master.

The communication data are exchanged between the USB/RS232 master and the gateway using the maxon-specific USB/RS232 protocol. The data between the gateway and the addressed device 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'!

Communication Examples

In order to clarify the behaviour of the communication via gateway this section shows simple examples for USB and RS232 communication. The process of reading a 32-bit parameter from a device in the CAN network is shown.

USB Example

Object:	DeviceType, Index 0x1000, Sub Index 0x00
Node:	2
USB Command:	ReadObject
CANopen Service:	SDO Upload (Expedited Transfer)

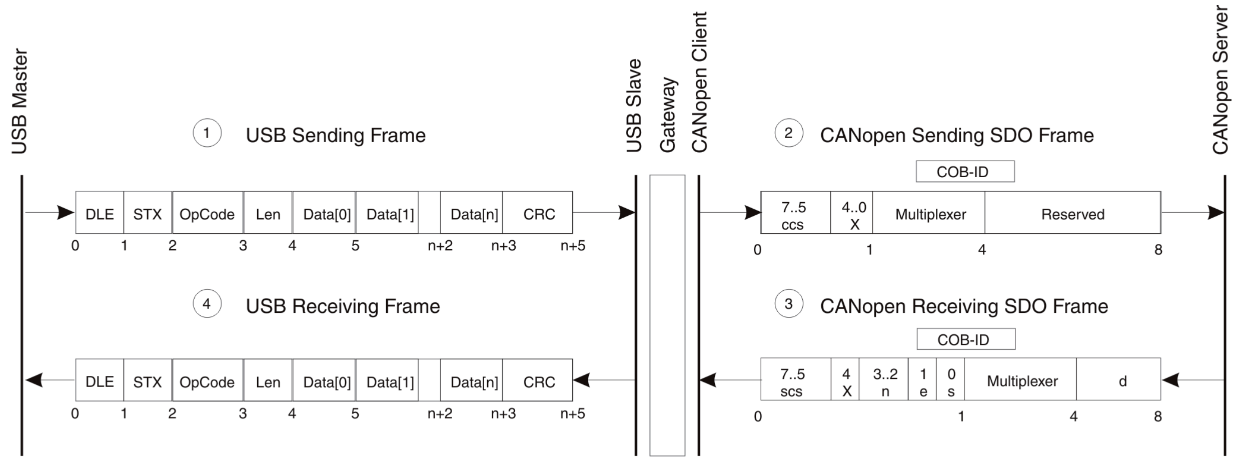

Figure 2: *Communication Example USB*

Step 1: USB Sending Frame		
DLE	0x90	Data Link Escape
STX	0x02	Start of Text
OpCode	0x10	ReadObject command
Len	0x02	2 Data Words
Data[0]	0x00	Index Byte 0
Data[1]	0x10	Index Byte 1
Data[2]	0x00	Sub Index
Data[3]	0x02	Node Id
CRC[0]	0xDF	Checksum Byte 0
CRC[1]	0xF2	Checksum Byte 1

Step 2: CANopen Sending SDO Frame		
COB-ID	0x602	0x600 + Node ID
Data[0]	0x40	ccs = 2
Data[1]	0x00	Index Byte 0
Data[2]	0x10	Index Byte 1
Data[3]	0x00	Sub Index
Data[4]	0x00	reserved
Data[5]	0x00	reserved
Data[6]	0x00	reserved
Data[7]	0x00	reserved

Step 3: CANopen Receiving SDO Frame		
COB-ID	0x582	0x580 + Node ID
Data[0]	0x43	scs = 2, n = 0, e = 1, s = 1
Data[1]	0x00	Index LowByte
Data[2]	0x10	Index HighByte
Data[3]	0x00	Sub Index
Data[4]	0x92	DeviceType Byte 1
Data[5]	0x01	DeviceType Byte 2
Data[6]	0x02	DeviceType Byte 3
Data[7]	0x00	DeviceType Byte 4

Step 4: USB Receiving Frame		
DLE	0x90	Data Link Escape
STX	0x02	Start of Text
OpCode	0x00	Answer to ReadObject
Len	0x04	4 Data Words
Data[0]	0x00	ErrorCode Byte 0
Data[1]	0x00	ErrorCode Byte 1
Data[2]	0x00	ErrorCode Byte 2
Data[3]	0x00	ErrorCode Byte 3
Data[4]	0x92	DeviceType Byte 0
Data[5]	0x01	DeviceType Byte 1
Data[6]	0x02	DeviceType Byte 2
Data[7]	0x00	DeviceType Byte 3
CRC[0]	0x9A	Checksum Byte 0
CRC[1]	0xED	Checksum Byte 1

Note: For description of ccs, scs, ... go to Legend.

RS232 Example

Object:	DeviceType, Index 0x1000, Sub Index 0x00
Node:	2
RS232 Command:	ReadObject
CANopen Service:	SDO Upload (Expedited Transfer)

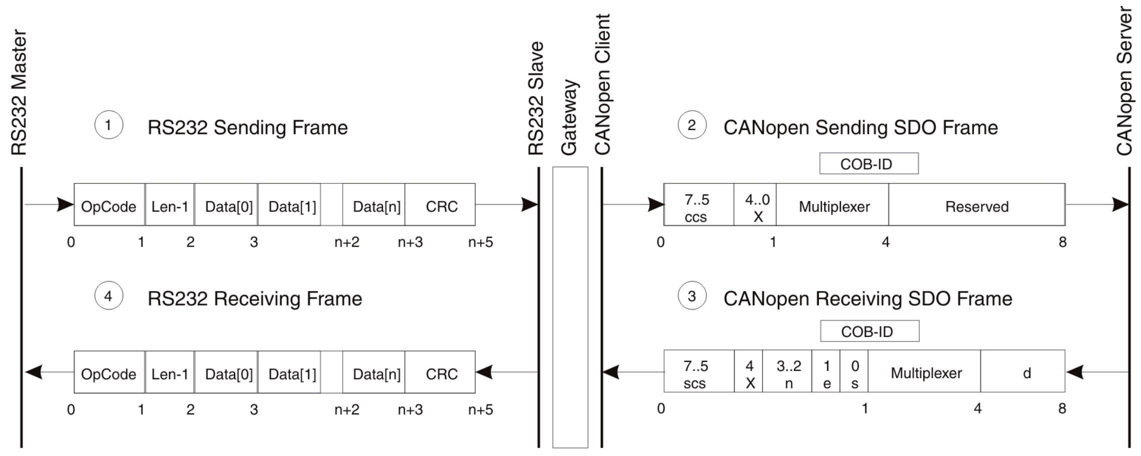
RS232 Master
RS232 Slave
Gateway
CANopen Client
CANopen Server
1 RS232 Sending Frame
OpCode Len-1 Data[0] Data[1] Data[n] CRC
0 1 2 3 n+2 n+3 n+5
4 RS232 Receiving Frame
2 CANopen Sending SDO Frame
COB-ID
7..5 ccs 4..0 X Multiplexer Reserved
0 1 4 8
3 CANopen Receiving SDO Frame
7..5 scs 4 X 3..2 n 1 e 0 s Multiplexer d

Figure 3: *Communication Example RS232*

Step 1: RS232 Sending Frame		
OpCode	0x10	ReadObject command
Len-1	0x01	2 Data Words
Data[0]	0x00	Index Byte 0
Data[1]	0x10	Index Byte 1
Data[2]	0x00	Sub Index
Data[3]	0x02	Node Id
CRC[0]	0x10	Checksum Byte 0
CRC[1]	0xCD	Checksum Byte 1

Step 2: CANopen Sending SDO Frame		
COB-ID	0x602	0x600 + Node Id
Data[0]	0x40	ccs = 2
Data[1]	0x00	Index Byte 0
Data[2]	0x10	Index Byte 1
Data[3]	0x00	Sub Index
Data[4]	0x00	reserved
Data[5]	0x00	reserved
Data[6]	0x00	reserved
Data[7]	0x00	reserved

Step 4: RS232 Receiving Frame		
OpCode	0x00	Answer to ReadObject
Len-1	0x03	4 Data Words
Data[0]	0x00	ErrorCode Byte 0
Data[1]	0x00	ErrorCode Byte 1
Data[2]	0x00	ErrorCode Byte 2
Data[3]	0x00	ErrorCode Byte 3
Data[4]	0x92	DeviceType Byte 0
Data[5]	0x01	DeviceType Byte 1
Data[6]	0x02	DeviceType Byte 2
Data[7]	0x00	DeviceType Byte 3
CRC[0]	0xEB	Checksum Byte 0
CRC[1]	0x6D	Checksum Byte 1

Step 3: CANopen Receiving SDO Frame		
COB-ID	0x582	0x580 + Node Id
Data[0]	0x43	scs = 2, n = 0, e = 1, s = 1
Data[1]	0x00	Index LowByte
Data[2]	0x10	Index HighByte
Data[3]	0x00	Sub Index
Data[4]	0x92	DeviceType Byte 0
Data[5]	0x01	DeviceType Byte 1
Data[6]	0x02	DeviceType Byte 2
Data[7]	0x00	DeviceType Byte 3

Legend:		
	ccs:	Client command specifier (Bit 7 ... 5)
	scs:	Server command specifier (Bit 7 ... 5)
	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 (Bit 3 ... 2).
	e:	Transfer type (0: normal transfer; 1: expedited transfer) (Bit 1).
	s:	Size indicator (0: data set size is not indicated; 1: data set size is indicated) (Bit 0).

Command Translation

The USB/RS232 command set is designed to be very close to the CANopen services. All USB/RS232 commands have a directly corresponding service on the CAN network. This approach helps to simplify the gateway functionality. No data has to be stored and buffered between two following USB/RS232 commands. The memory use of the gateway can be minimized. All received data are directly forwarded to the CAN bus.

USB/RS232 command		CANopen service
ReadObject	⇨	Initiate SDO Upload / Expedited Transfer
InitiateSegmentedRead	⇨	Initiate SDO Upload / Normal Transfer
SegmentRead	⇨	Upload SDO Segment
WriteObject	⇨	Initiate SDO Download / Expedited Transfer
InitiateSegmentedWrite	⇨	Initiate SDO Download / Normal Transfer
SegmentWrite	⇨	Download SDO Segment
SendNMTService	⇨	NMT Service
ReadLSSFrame	⇨	LSS Service
SendLSSFrame	⇨	LSS Service

This type of implementation gives a greater responsibility to the programmer of the USB/RS232 protocol. The programmer has to decide whether to use the non-segmented commands or the segmented commands. This decision has to be made considering the number of data bytes to be transmitted. The non-segmented commands (ReadObject, WriteObject) support only up to 4 data bytes. More than 4 data bytes have to be transmitted using the segmented commands (InitiateSegmentedRead, SegmentRead, InitiateSegmentedWrite, SegmentWrite).

Limiting Factors

The number of segments has a big influence on the performance of the data exchange. Exchanging data directly with a device connected to RS232 (no gateway), a data segment can transfer up to 63 Bytes per command. This means that for 1kB of data, 17 commands have to be sent. In comparison, sending this data to a device to be addressed via gateway, 147 commands have to be sent. The CANopen services (normal transfer) allow only 7 bytes to be transferred in a segment. So the CANopen segment limits also the RS232 segment. Remember: the gateway is not capable to buffer data and to split the data into several CANopen services.

Considering the segment size, the CANopen is the limiting factor for the communication performance. Considering the bit rate of the two field buses, the RS232 is the limiting factor. The communication via gateway can not take the advantage of the high bit rate of the CAN bus. The communication via gateway is limited by the slow bit rate of RS232 and the small segment size of CANopen.

	USB protocol	RS232 protocol	CANopen	USB to CANopen gateway	RS232 to CANopen gateway
Max. bit rate	1 MBit/s	115.2 kBit/s	1 MBit/s	1 MBit/s	115.2 kBit/s
Max. segment size	63 Bytes	63 Bytes	7 Bytes	7 Bytes	7 Bytes
Conclusion					
Transfer Rate Segment Size	Fast Big	Slow Big	Fast Small	Fast Small	Slow Small

Table 1: *Limiting Factors*

However, these limiting factors have to be put in perspective, because most of the objects in the object dictionary are 32-bit parameters or even smaller. So segmented transfer is used very rarely. Only for reading the data buffer of the data recorder or for a firmware download, has the segmented transfer to be used.

Timing RS232

The primary bottleneck of the communication via RS232 to CANopen gateway is the RS232 bit rate. The maximum RS232 bit rate of 115.2 kBit/s is ten times smaller than the maximum CAN bit rate of 1 MBit/s. The duration of the communication depends more or less on the RS232 bit rate used. The timing example below shows the delaying of the communication for addressing a device via the gateway.

Timing Example

Test Platform	Pentium 4, 2.66 GHz, Windows XP, EPOS_UserInterface
Command	ReadObject, 32-Bit Object
RS232 Bit rate	38400 Bit/s (Default)
CAN Bit rate	1 MBit/s (Default)
Time via Gateway	10.125 ms (measured)
Time without Gateway	9.995 ms (measured)
Delay	130 µs

The delay time without gateway is relatively short in relation to the whole time and can be ignored.

Timing Values

CAN	**Read/Write 8-bit / 16-bit / 32-bit object (2 CAN frames @ 8/8 bytes)**		**Read/Write (200 CAN frames @ 8/8 bytes)**	
Bit rate	Calculated	Measured*	Calculated	Measured*
1 MBit/s	220 µs	794 µs	44 ms	159 ms
800 kBit/s	275 µs	850 µs	55 ms	170 ms
500 kBit/s	440 µs	1.0 ms	88 ms	204 ms
250 kBit/s	880 µs	1.5 ms	196 ms	307 ms
125 kBit/s	1.8 ms	2.4 ms	360 ms	488 ms
50 kBit/s	4.4 ms	5.3 ms	880 ms	1052 ms
20 kBit/s	11 ms	12.4 ms	2.2 s	2.5 s

Table 2: *Timing CAN Bus (CANopen SDO services)*

USB	**Read/Write 8-bit / 16-bit / 32-bit object (2 USB frames @ 10/14 bytes)**		**Read/Write (200 USB frames @ 10/14 bytes)**	
Bit rate	Calculated	Measured	Calculated	Measured
1 MBit/s	2 ms	2.3 ms	400 ms	474 ms

Table 3: *Timing USB*

RS232	**Read/Write 8-bit / 16-bit / 32-bit object (2 RS232 frames @ 10/14 bytes)**		**Read/Write (200 RS232 frames @ 8/18 bytes)**	
Bit rate	Calculated	Measured	Calculated	Measured
115200 Bit/s	2.083 ms	3.9 ms	0.42 s	0.8 s
57600 Bit/s	4.16 ms	7.2 ms	0.83 s	1.4 s
38400 Bit/s	6.25 ms	10.4 ms	1.25 s	2.1 s
19200 Bit/s	12.5 ms	20.5 ms	2.5 s	4.1 s
14400 Bit/s	16.6 ms	27.2 ms	3.33 s	5.5 s
9600 Bit/s	34.47 ms	40.7 ms	6.89 s	8.2 s

Table 4: *Timing RS232 (maxon specific protocol)*

* Measured PC using IXXAT card with driver VCI3

Conclusion

The gateway functionality enables an easy connection to the CAN network. The user does not need a separate CAN interface card for monitoring a CAN network. Also, the wiring of the CAN network does not have to be altered. By just plugging in the USB cable or RS232 cable to one of the EPOS2 positioning controllers, the user is able to control and monitor all EPOS2 devices in the network.

The communication delay for the CAN communication can be neglected comparing to the duration for the RS232 baud rate. This means the gateway does not slow down the RS232 communication. Thereby, it does not make any difference whether the master is addressing a device directly via RS232 or via the gateway in the CAN network (Exception: segmented transfers).