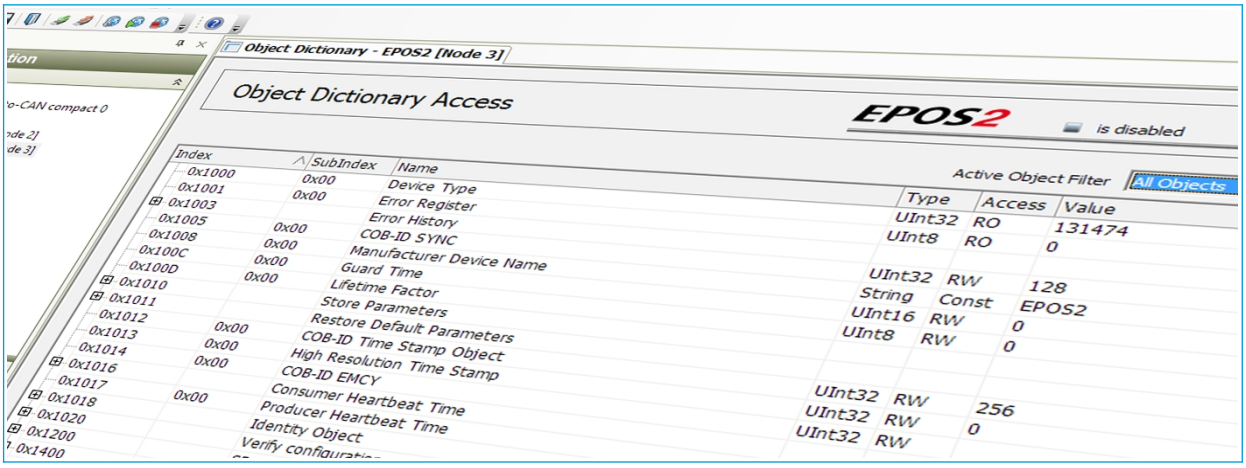
neMESYS Syringe Pump

Firmware Specification



ORIGINAL MANUAL - SEPTEMBER 2021

CETONI GmbH
Wiesenring 6
07554 Korbussen
Germany

T +49 (0) 36602 338-0
F +49 (0) 36602 338-11
E info@cetoni.de

www.cetoni.de

1 Summaries and directories

1.1 Table of contents

1.2 Change history

REVISION	CHANGE
28.11.2006	Creation of document
18.12.2006	Added neMESYS RS232 Library documentation
16.02.2014	Removed RS232 Library command reference because it is already in file neMESYS_RS232_Library.chm
12.02.2016	Added Pump Control chapter for a detailed description of common neMESYS pump control tasks
11.12.2019	Added example frames for RS232 protocol Added documentation for reading analog input values
05.07.2021	Added Development Tools section
14.09.2021	Improved valve switching documentation

2 About this Document

2.1 Intended Purpose

The purpose of the present document is to familiarize you with the described equipment and the tasks on safe and adequate installation and/or commissioning. Observing the described instructions in this document will help you:

- to avoid dangerous situations,
- to keep installation and/or commissioning time at a minimum and
- to increase reliability and service life of the described equipment.

Use for other and/or additional purposes is not permitted. cetoni, the manufacturer of the equipment described, does not assume any liability for loss or damage that may arise from any other and/or additional use than the intended purpose.

2.2 Target Audience

This document is meant for trained and skilled personnel working with the equipment described. It conveys information on how to understand and fulfill the respective work and duties. This document is a reference book. It does require particular knowledge and expertise specific to the equipment described.

2.3 Symbols and Signal Words Used

The following symbols are used in this manual and are designed to aid your navigation through this document:

HINT. Describes practical tips and useful information to facilitate the handling of the software.

IMPORTANT. Describes important information and other especially useful notes, in which no dangerous or damaging situations can arise.

ATTENTION. Indicates a potentially damaging situation. Failure to avoid this situation may result in damage to the product or anything nearby.

CAUTION. Describes a situation that may be dangerous. If this aspect is not avoided, light or minor injuries as well as damage to property could result.

3 System Overview

3.1 General Device Architecture

The device implements a CANopen slave device. CANopen is the internationally standardized (EN 50325-4) higher-layer protocol for embedded control system. The set of CANopen specification comprises the application layer and communication profile as well as application, device, and interface profiles. CANopen provides very flexible configuration capabilities. These specifications are developed and maintained by CiA members.

The communication interface of the device follows the CiA CANopen specifications as follows:

- *CiA 301* – Application Layer and Communication Profile
- *CiA 306* – Electronic Data Sheet Specification
- *CiA 303-2* – Representation of SI units and prefixes
- *CiA 303-3* – Indicator Specification

A CANopen device can be logically structured in three parts.

One part provides the communication interface (CAN, RS232) and another part provides the device's application, which controls e.g. the Input/Output (I/O) lines of the device in case of an I/O module.

The interface between the application and the CAN-interface is implemented in the object dictionary. The object dictionary is unique for any CANopen device. It is comparable to a parameter list and offers the access to the supported configuration- and process data.

The following section explains the basic concepts related to the CANopen protocol application layer. This document is intended as a basic overview only, and users are encouraged to review the CiA DS 301 specification for more information.

3.2 Object Dictionary

The most significant part of any CANopen device is the Object Dictionary. It is essentially a grouping of

objects accessible via the network (via CAN or RS232) in an ordered, predefined fashion. The object dictionary is essentially a table, that stores configuration and process data. The figure below shows an example of an object dictionary. Each object within the dictionary is addressed using a 16-bit index 1 and an 8-bit subindex 2.

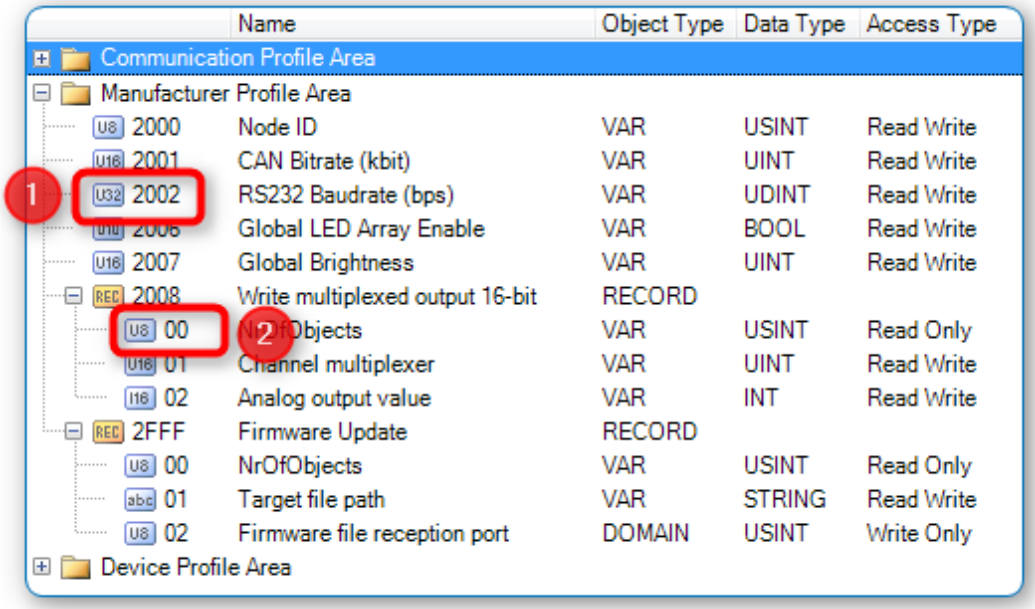


Figure 1: Object dictionary example

The 16-bit index 1 is used to address all entries within the Object Dictionary. In case of a simple variable, it references the value of this variable directly. In case of records and arrays however, the index addresses the entire data structure. The subindex 2 permits individual elements of a data structure to be accessed via the network.

- For single Object Dictionary entries (such as UNSIGNED8, BOOLEAN, INTEGER32, etc.), the subindex value is always zero.
- For complex Object Dictionary entries (such as arrays or records with multiple data fields), the subindex references fields within a data structure pointed to by the main index. This allows for up to 255 sub-entries at each index. Each entry can be variable in type and length.

The overall layout of the standard Object Diction-ary conforms to other industrial field bus concepts.

Index	Description
0000_h	Reserved

0001_h-$009F_h$	Data types (not supported)
$00A0_h$-$0FFF_h$	Reserved
1000_h-$1FFF_h$	Communication Profile Area (CiA 301)
2000_h-$5FFF_h$	Manufacturer-specific Profile Area
6000_h-$9FFF_h$	Standardized Device Area (e.g. CiA 401 – I/O Modules)
$A000_h$-$FFFF_h$	Reserved

Table 1: Object dictionary layout

Access to each object dictionary entry is possible via SDO transfer (CAN) or via RS232 protocol by simply providing the index and sub index of the object dictionary entry to access.

4 CAN Communication

4.1 Introduction

This chapter provides general information about CAN communication and CANopen application layer. The information is relevant only for devices that support CAN communication via CAN interface. If your device only supports serial communication via RS232 CANopen Serial Interface (CSI), you can skip this chapter.

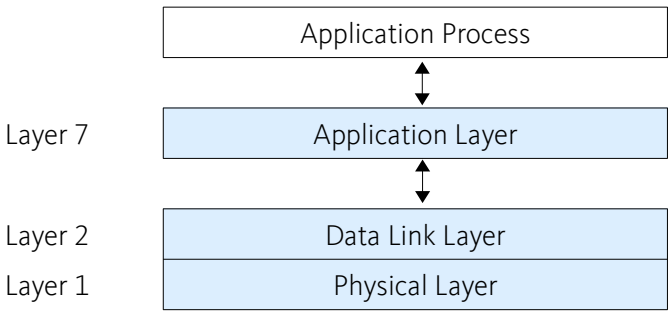
HINT. An excellent and easy to understand introduction to CAN and CANopen is available here:
http://www.canopensolutions.com/english/about_canopen/about_canopen.shtml

HINT. You can skip this chapter if your device does not support communication via CAN interface.

4.2 Reference Model of Data Communication

The Open Systems Interconnection Reference Model (OSI Reference Model) forms the basis for the description of communication systems today. The OSI model describes data communication systems in the form of a layer model, consisting of seven different layers, and assigns specific services to each layer. Simpler communication systems do not require all the functionalities of the OSI model. In general, only three layers (physical layer, data link layer and application layer) are relevant for data communication in the automation area.

The three layers shown in the figure implement the most important tasks of data communication in the fieldbus area.



4.3 CAN-Bus

4.3.1 CAN in the OSI reference model

The CAN protocol was specified by the company BOSCH. Regarding the OSI reference model, the CAN specification implements the data link layer completely and the physical layer partially. The physical signal representation is defined in the CAN protocol, while the form of the bus medium and the bus coupling was not specified.

4.3.2 Bus topology and data rate

The CAN bus uses a linear bus topology. The number of nodes is not limited by the CAN protocol, but depends on the performance of the driver chips used. Data rates up to 1 Mbit/s (network extension up to 40 m) and network extensions up to 1,000 m (at 80 Kbit/s) are possible. Two-wire lines with differential levels as well as fibre optic cables are possible as transmission medium.

4.3.3 Message transfer

The message receiver is not addressed, but the CAN messages are identified by a unique identifier – the CAN ID. Message transmission is based on the producer-consumer principle. This means that a message sent by one CAN node (producer) can be received by all other CAN nodes (consumers). On the basis of the message identifier, a subscriber decides whether a message is relevant for him or not.

4.3.4 Bus access

The identifier of a CAN message determines its priority. The message with the lowest CAN ID has the

highest priority. Each message ID may only be sent from one CAN node in the system to avoid collisions. If several CAN nodes start sending a message at the same time, a collision occurs. This conflict is resolved by giving the message with the highest priority (with the lowest ID) bus access.

If the message with the highest priority has been sent, bus arbitration starts again for the remaining messages until all messages have been sent. This ensures that messages are not destroyed or lost.

4.3.5 Length of the payload data

The maximum data length of a CAN message is limited to 8 bytes. This enables fully functional data transmission in very difficult electromagnetic environments and guarantees short latency times for bus access of high priority messages.

4.3.6 Structure of CAN Frames

The CAN specification distinguishes between two compatible message formats, the standard format with 11 bit identifier and the extended format with 29 bit identifier. CETONI devices only use messages with 11-bit identifiers. A CAN message in standard format is shown in figure below and consists of:

- 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 – the Control Field – 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 Acknowledgment (**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 returning a dominant bit.
- The recessive bits of the End of Frame (**EOF**) terminate the Data Frame. Between two frames, a

recessive 3-bit Intermission field must be present.

CETONI devices only use messages with 11-bit identifiers:

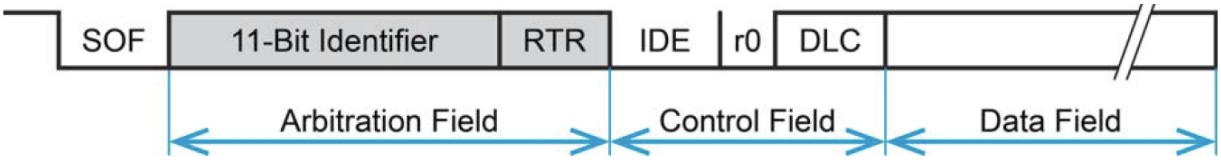


Figure 2: Standard frame format

- The Identifier's (COB-ID) length in the Standard Format is 11 bit.
- The Identifier is followed by the RTR (Remote Transmission Request) bit. In Data Frames, the RTR bit must be dominant, within a Remote Frame, the RTR bit must 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 must be sent dominant, but receivers accept dominant and recessive bits in all combinations.

4.3.7 Error Checking and Fault Confinement

The robustness of CAN may be attributed in part to its abundant error-checking procedures. The CAN protocol incorporates five methods of error checking: three at the message level and two at the bit level. If a message fails any one of these error detection methods, it is not accepted and an error frame is generated from the receiving node. This forces the transmitting node to resend the message until it is received correctly. However, if a faulty node hangs up a bus by continuously repeating an error, its transmit capability is removed by its controller after an error limit is reached. The following methods for error detection are used:

- Error checking at the message level is enforced by the CRC and the ACK slots. The 16-bit CRC contains the checksum of the preceding application data for error detection with a 15-bit checksum and 1-bit delimiter. The ACK field is two bits long and consists of the acknowledge bit and an acknowledge delimiter bit.
- Also at the message level is a form check. This check looks for fields in the message which must

always be recessive bits. If a dominant bit is detected, an error is generated. The bits checked are the SOF, EOF, ACK delimiter, and the CRC delimiter bits

- At the bit level, each bit transmitted is monitored by the transmitter of the message. If a data bit (not arbitration bit) is written onto the bus and its opposite is read, an error is generated. The only exceptions to this are with the message identifier field which is used for arbitration, and the acknowledge slot which requires a recessive bit to be overwritten by a dominant bit.
- The final method of error detection is with the bit-stuffing rule where after five consecutive bits of the same logic level, if the next bit is not a complement, an error is generated.

CAN uses the principle of error signalling. Detected errors are reported to the other network users by sending an error frame. This ensures that the communication with all functioning CAN nodes of a network is error-free and consistent and guarantees very short error response times.

4.4 CANopen Basics

4.4.1 Introduction

CANopen is a standardized application for distributed automation systems based on CAN (Controller Area Network) offering the following performance features:

- Transmission of time-critical process data according to the producer consumer principle
- Standardized device description (data, parameters, functions, programs) in the form of the so-called "object dictionary". Access to all "objects" of a device with standardized transmission protocol according to the client-server principle
- Standardized services for device monitoring (node guarding/heartbeat), error signalisation (emergency messages) and network coordination ("network management")
- Standardized system services for synchronous operations (synchronization message), central time stamp message
- Standardized help functions for configuring baud rate and device identification number via the bus
- Standardized assignment pattern for message identifiers for simple system configurations in the form of the so-called "predefined connection set"

Subsequently described are the CANopen communication features most relevant to the CETONI CANopen devices. For more detailed information consult above mentioned 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

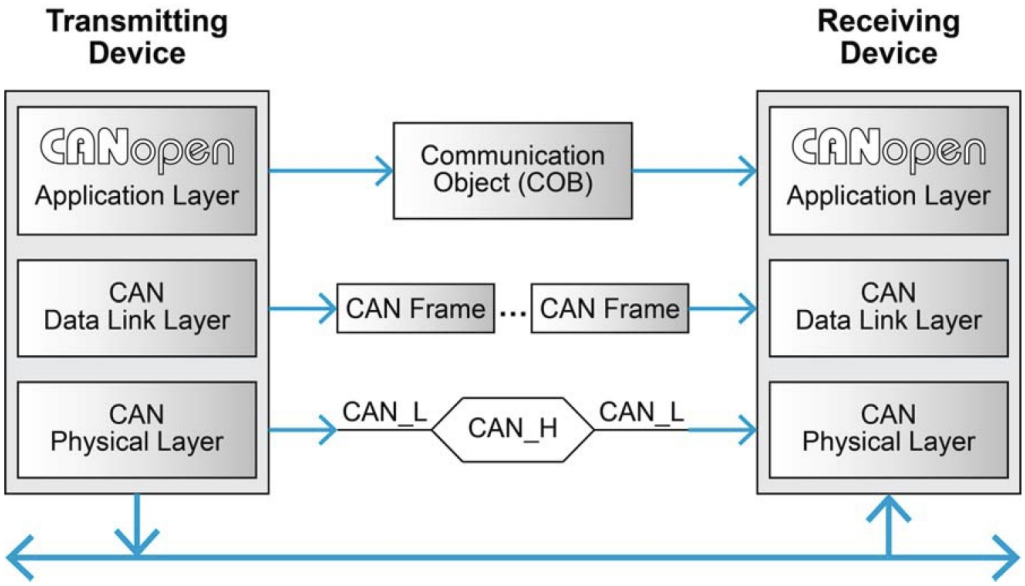


Figure 3: Protocol Layer Interactions

4.4.2 Physical Structure of the CAN Network

CANopen is a networking system based on the CAN serial bus. It assumes that the device's hardware features a CAN transceiver and a CAN controller as specified in ISO 11898. The physical medium is a differently driven 2-wire bus line with common return. The underlying CAN architecture defines the basic physical structure of the CANopen network. Therefore, a line (bus) topology is used. To avoid reflections of the signals, both ends of the network must be terminated. In addition, the maximum permissible branch line lengths for connection of the individual network nodes are to be observed.

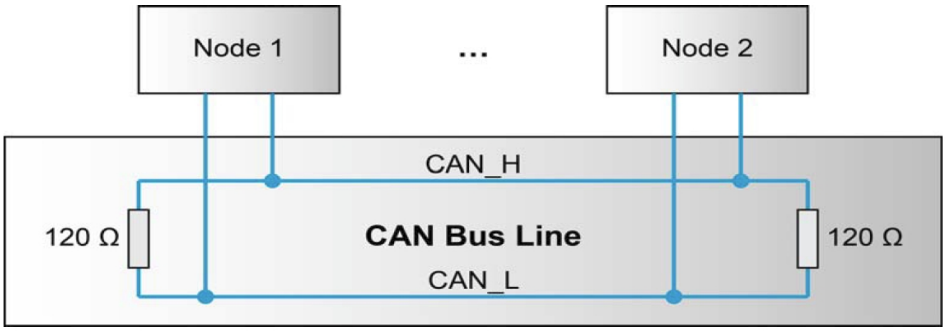


Figure 4: ISO 11898 basic network setup

The recommended permissible bit rates for a CANopen network are given in CiA 301: 10 kbps, 20 kbps, 50 kbps, 125 kbps, 250 kbps, 500 kbps, 800 kbps and 1000 kbps. In CiA 301 a recommendation for the configuration of the bit timing is also given.

Additionally, for CANopen, two additional conditions must be fulfilled:

- All nodes must be configured to the same bit rate and
- No node-ID may exist twice.

4.5 Communication Objects

CANopen uses communication objects for data transmission in the network. The following communication objects are specified by CANopen:

- Service data objects (**SDO**) are used to access the entries in the object dictionary.
- Process data objects (**PDO**) are used for fast transmission of process information
- Objects with special functions provide various system services (synchronization objects, time service objects, emergency objects)
- Network management objects (**NMT**) are necessary to start, stop and monitoring of network participants

In a CAN network, all objects refer to a specific message identifier. This means that each communication object has a unique CAN ID, and certain CAN message IDs are reserved for certain objects.

4.5.1 Service Data Objects – SDOs

With Service Data Objects (SDOs), the access to entries of a device Object Dictionary is provided. A SDO is mapped to two CAN Data Frames with different identifiers, because 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.

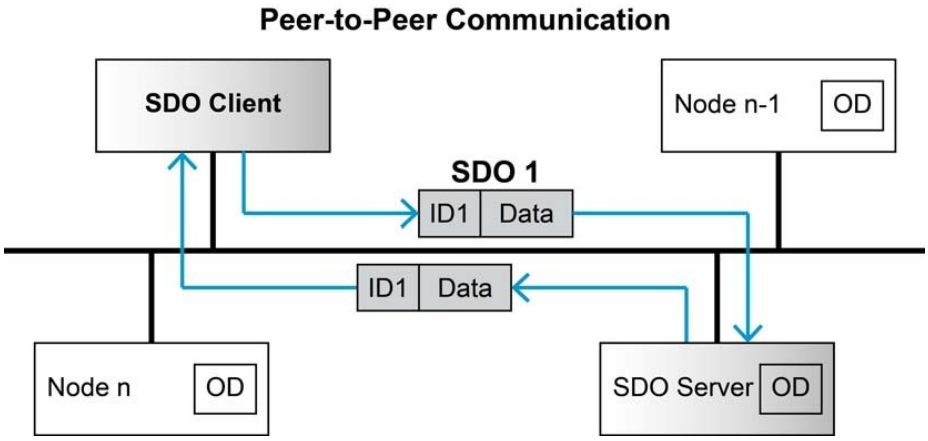


Abbildung 4.1: Service Data Object (SDO)

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 a CANopen network, up to 256 SDO channels requiring two CAN identifiers each may be used.

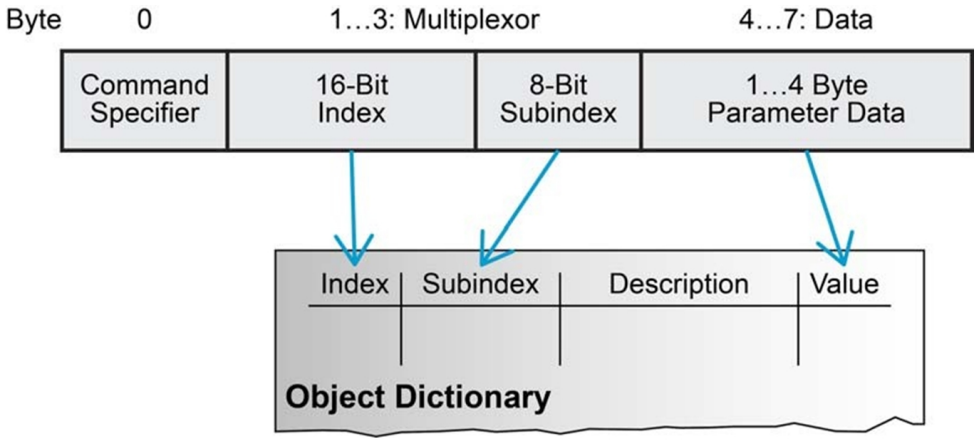


Abbildung 4.2: Object Dictionary Access

4.5.2 Process Data Objects – PDOs

Process data represents data that can be changing in time, such as the inputs (i.e. sensors) and outputs (i.e. motor drives) of the node controller. Process data is also stored in the object dictionary. However, since SDO communication only allows access to one object dictionary index at a time, there can be a lot of overhead for accessing continually changing data. In addition, the CANopen protocol has the requirement that a node must be able to send its own data, without needing to be polled by the CANopen master. Thus, a different method is used to transfer process data, using a communication method called Process Data Objects (**PDO**s).

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 numerous other devices (broadcasting). PDOs are transmitted in a non-confirmed mode. The producer sends a Transmit PDO (TxPDO) with a specific identifier that corresponds to the identifier of the Receive PDO (RxPDO) of one or more consumers.

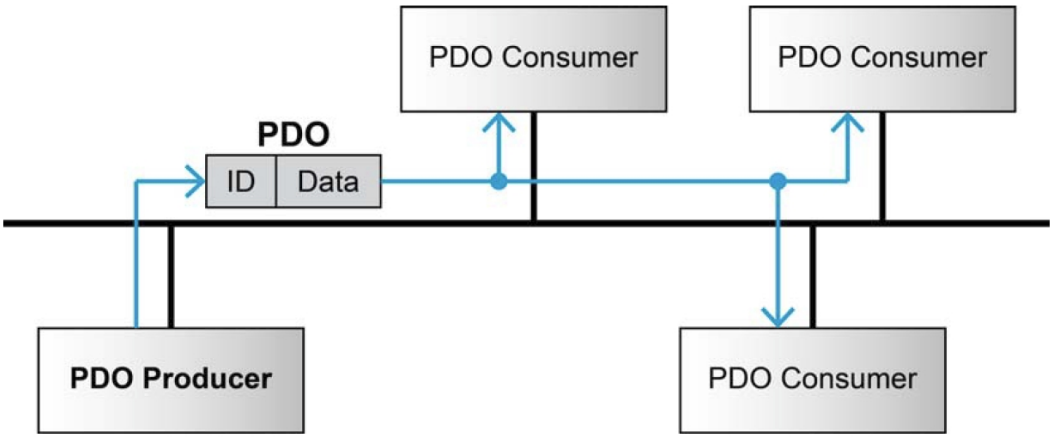


Figure 5: Process Data Object (PDO)

There are two types of PDOs: transfer PDOs (TPDOs) and receive PDOs (RPDOs). A TPDO is the data coming from the node (produced) and a RPDO is the data coming to the node (consumed). In addition, there are two types of parameters for a PDO: the configuration parameters and the mapping parameters. The section of the object dictionary reserved for PDO configuration and mapping information are indices 1400h-1BFFh.

IMPORTANT. PDO communication is not permitted in NMT state Pre-Operational. Switch to NMT Operational state to enable PDO transmission.

4.5.2.1 PDO CONFIGURATION PARAMETERS

The configuration parameters specify the COB-ID, the transmission type, inhibit time (TPDO only) and the event timer, which are explained in this section. There are different methods through which a PDO transfer can be initiated. These methods include event driven, time driven, individual polling and synchronized polling. The type of transmission is specified in the configuration parameters of the PDO. In event driven transmission, the PDO transfer is initiated when the process data in it changes.

- In time driven transmission, the PDO transfer occurs at a fixed time interval.
- In event-driven transmission a PDO transfer is triggered by the occurrence of an object-specific event or change of process data
- In individual polling, the PDO transfer is initiated using a mechanism called remote request, which is not commonly used.
- In synchronized polling, the PDO transfer is initiated using a SYNC signal. The sync signal is

frequently used as a global timer. For example, if the CANopen master sends out a SYNC message, multiple nodes may be configured to see and respond to that SYNC. In this way, the master is able to get a "snapshot" of multiple process objects at the same time.

4.5.2.2 PDO MAPPING PARAMETERS

The mapping parameters specify which object dictionary values are sent by a single PDO message. For example, a single PDO message may contain data from object index 2001h, 2003h and 2005h.

Index	SubIndex	Example Value	Description
1801h	00h	100	Highest Subindex
	01h	00000181h	PDO COB ID
	02h	0	
	03h	1000	Inhibit Time
	04h	Unused	Unused
	05h	0 (disabled)	Event timer
1A01h	00h	3	Number of Entries
	01h	Index 2001h, Subindex 00h	Mapped OD item 1
	02h	Index 2003h, Subindex 00h	Mapped OD item 2
	03h	Index 2005h, Subindex 00h	Mapped OD item 3

Figure 6: TPDO 1 Communication Parameters (0x1801h) and Mapping Parameters (0x1A01h)

4.5.3 Sync Object

The SYNC producer provides the synchronization signal for the SYNC consumer.

As the SYNC consumers receive the signal, they will commence carrying out their synchronous tasks. In general, fixing of the transmission time of synchronous PDO messages coupled with the periodicity of the SYNC Object's transmission guarantees that sensors may arrange sampling of process variables and that actuators may apply their actuation in a coordinated manner. The identifier of the SYNC Object is available at index "1005h".

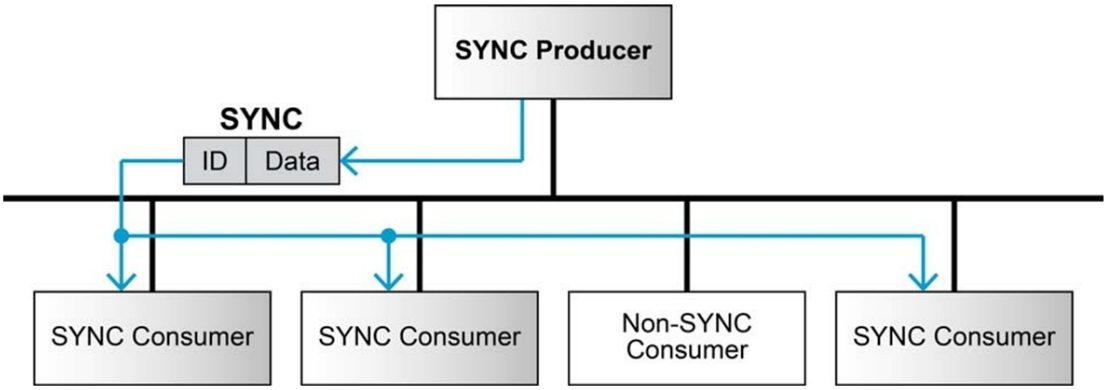


Figure 7

Figure 4.3: Synchronization Object (SYNC)

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 the most, once for every period of the SYNC. The time period between 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. Hence, they may also be transmitted within the synchronous window.

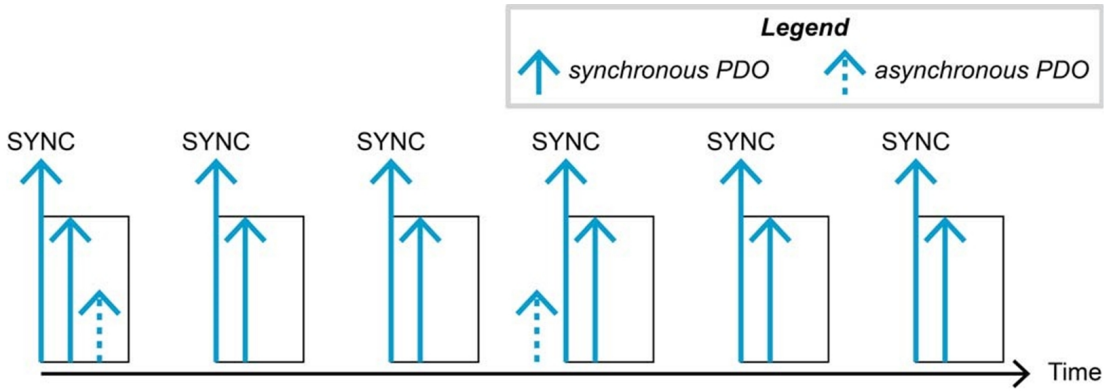


Figure 8: Synchronous PDO

4.6 Network Management – NMT

In addition to providing services and protocols for the transmission of process data and the configuration of devices, the operation of a system distributed over a network requires functions for the command control of the communication state of the individual network nodes. As data transmission by CANopen devices is in many cases event-controlled, continual monitoring of the communication ability of the network nodes is also required. CANopen provides so-called "network management" services and protocols for these tasks, namely:

- control of the communication state of network nodes and
- node monitoring.

4.6.1 NMT Services

The CANopen network management is node-oriented and follows a master/slave structure. It requires one device in the network that fulfils the function of the NMT Master. The other nodes are NMT Slaves.

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 nodes' and network's communication status.
- Configuration Control Services for up/downloading of configuration data from/to a network module.

A NMT Slave represents that part of a node, which is responsible for the node's NMT functionality. It is uniquely identified by its module ID.

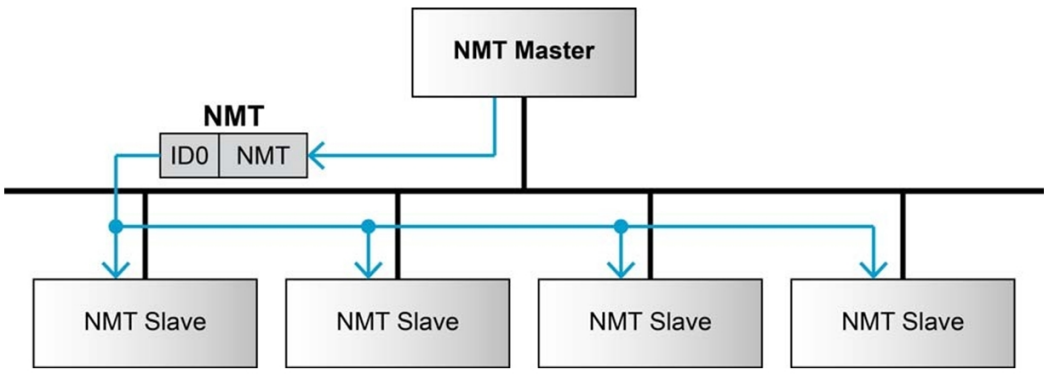


Figure 9: Network management (NMT)

The CANopen NMT Slave devices implement a state machine that automatically brings every device to “Pre-Operational” state, once powered and initialized.

The “**Pre-Operational**” state is primarily used for the configuration of CANopen devices. Therefore exchange of process data (via PDOs) is not possible in this state. The entries of the device object dictionaries can be accessed via "service data objects" (SDOs). By transmitting an SDO message, the object dictionary of a certain device can be modified, e.g. with a configuration tool.

IMPORTANT. PDO communication is not permitted in Pre-Operational state. Switch to Operational state to enable PDO transmission.

In addition to communication via SDO messages, emergency, synchronization, time stamp and of course NMT control messages can also be transmitted or received in the Pre-operational state. By transmitting a "Start-Remote-Node", a node switches to the "Operational" state.

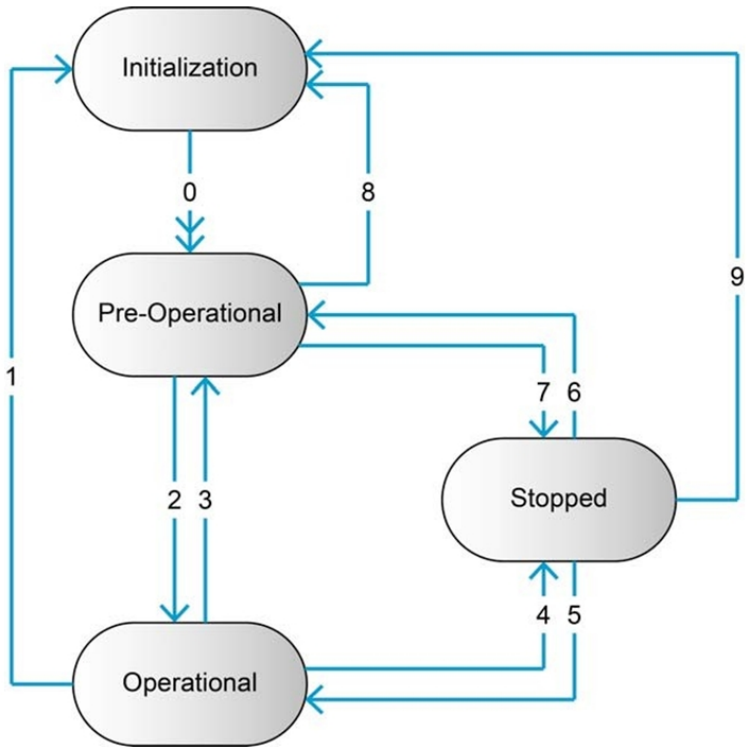


Figure 10: NMT slave states

In “**Operational**” state, PDO transfer is permitted. Furthermore, “Operational” can be used to achieve certain application behavior. The behavior's definition is part of the device profile’s scope. In “Operational”, all communication objects are active. Object Dictionary access via SDO is possible. However, implementation aspects or the application state machine may require to switching off or to

read only certain application objects while being operational (e.g. an object may contain the application program, which cannot be changed during execution).

By switching a device into "Stopped" state it will be forced to stop PDO and SDO communication. Except for node guarding or heartbeat messages, a node cannot transmit or receive any other messages in this state.

4.7 CANopen Error Handling – EMCY

4.7.1 Principle

Emergency objects are triggered by the occurrence of a CANopen device internal error situation and are transmitted from an emergency producer on the CANopen device. They are assigned the highest possible priority to ensure that they get access to the bus without latency. Emergency objects are suitable for interrupt type error alerts. An emergency object is transmitted only once per 'error event'. No further emergency objects will be transmitted as long as no new errors occur on a CANopen device.

Zero or more emergency consumers may receive the emergency object.

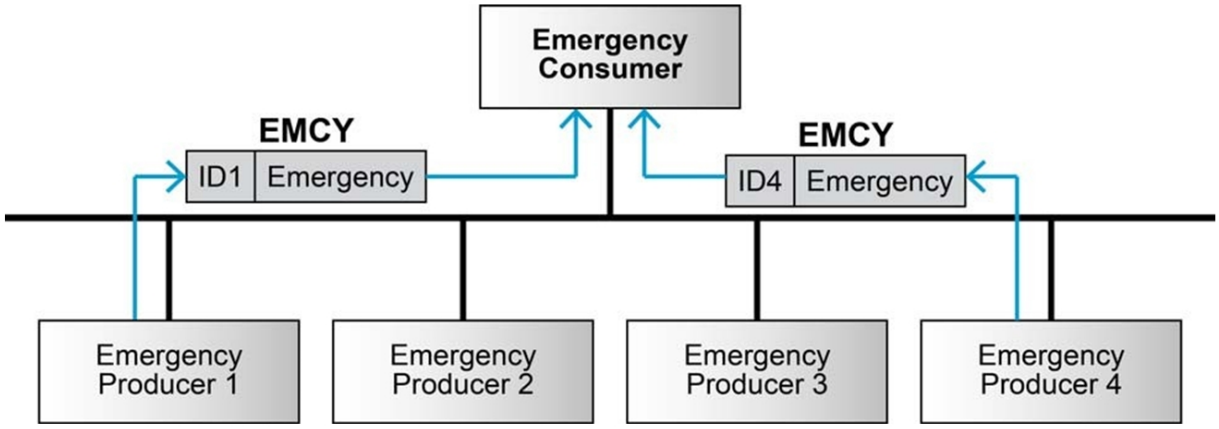


Figure 11: Emergency service (EMCY)

Simultaneously with transmission of the emergency message, the device writes the error code to [1003], where the error history is stored. The error register is content of the OD entry [1001] with bit-wise coding of the error cause

4.7.2 Emergency Message Frame

The device transmits emergency message frames over the CANopen network using *COB-ID EMCY (H1014)*. An emergency message consists of the error code with pre-defined error numbers and the

actual state of the *Error Register (H1001)*.

Byte	0	1	2	3	4	5	6	7
Description	Error Code		Error Register	Manufacturer specific error code				

Table 2: Emergency Message Frame

IMPORTANT. Emergency messages are only available for CAN bus communication and not for serial RS232 communication.

5 CANopen Serial Interface (CSI)

5.1 Overview

This section describes the cetoni CANopen Serial Interface (CSI). This is a serial protocol that enables access to CANopen device object dictionaries via a RS232 serial interface. The CANopen Serial Interface (CSI) supports an *Industrial RS232 Protocol with CRC checksum* for reliable RS232 connection of cetoni devices to control systems in industrial or laboratory environments. For a high degree of reliability in an electrically noisy environment, it features a checksum.

IMPORTANT. The protocol is a binary protocol with CRC checksum and handshaking. So it is not possible to simply access device parameters via serial terminal program.

5.2 Physical Layer

5.2.1 Electrical Standard

The CSI 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 device 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'.

5.2.2 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 Industrial RS232 Protocol with CRC checksum

6.1 Introduction

The serial EIA RS232 communication protocol is used to transmit and receive data over the cetoni device's RS232 serial port. Its principal task is to transmit data from a master (PC or any other central processing unit) to a single slave. The protocol is defined or point-to-point communication based on the EIA-RS232 standard.

The protocol can be used to implement the command set defined for cetoni devices. For a high degree of reliability in an electrically noisy environment, it features a checksum.

6.2 Protocol and Flow Control

6.2.1 Sequence of sending commands

The cetoni CANopen devices always communicates as a slave. A frame is only sent as an answer to a request. Some commands send an answer, other commands do not (observe respective descriptions to determine command that send an answer packets). The master always must start the communication by sending a packet structure.

Below described are the data flow while transmitting and receiving frame.

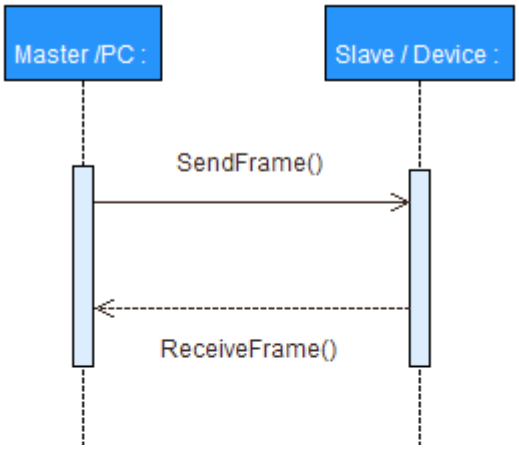


Figure 12: RS232 Communication – Command Sequence

6.2.2 Sending a data frame

When sending a frame, you will need to wait for different acknowledgment.

- After sending the first frame byte (*OpCode*), you will need to wait for the device's "*Ready Acknowledge*".
- Once the character "*O*" (okay) is received, the slave is ready to receive further data.
- If the character "*F*" (failed) is received, the slave is not ready to send data and communication must be stopped.
- After sending the checksum, you will need to wait for the "*End Acknowledge*". The slave sends either "*O*" (okay) or "*F*" (failed).

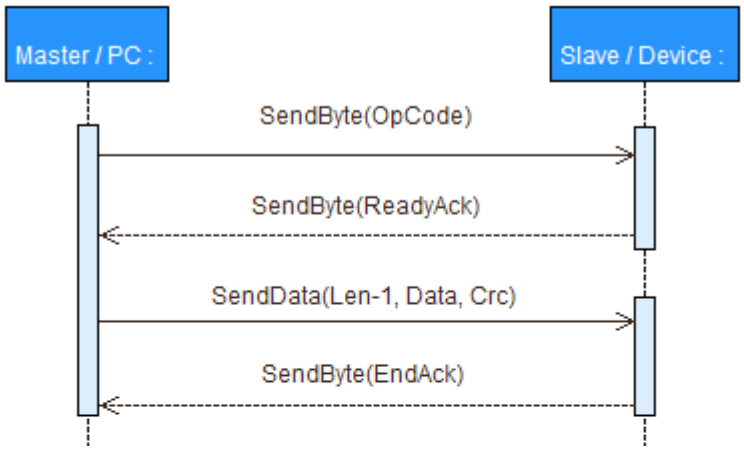


Figure 13: RS232 Communication – Sending a Data Frame to device

6.2.3 Receiving a data frame

In response to some of the command frames, the slave device returns a response data frame to the master. The data flow sequence is identical as for sending a data packet, only in the other direction. The master must also send the two acknowledges to the slave.

- The value of the first field must always be *0x00*, thus representing the operation code describing a response frame.
- After receiving the first byte, the master then must send the "*Ready Acknowledge*".
- Send character "*O*" (okay) if you are ready to receive the rest of the frame.
- Send character "*F*" (failed) if you are not ready to receive the rest of the frame.
- If the device does not get an "*O*" within the specified timeout, the communication is reset. Sending "*F*" does not reset the communication.
- After sending the "*Ready Acknowledge*" ("*O*"), the device sends the rest of the data frame. Then the checksum must be calculated and compared with the one received. If the checksum is correct, send acknowledge "*O*" to the device, otherwise send acknowledge "*F*".

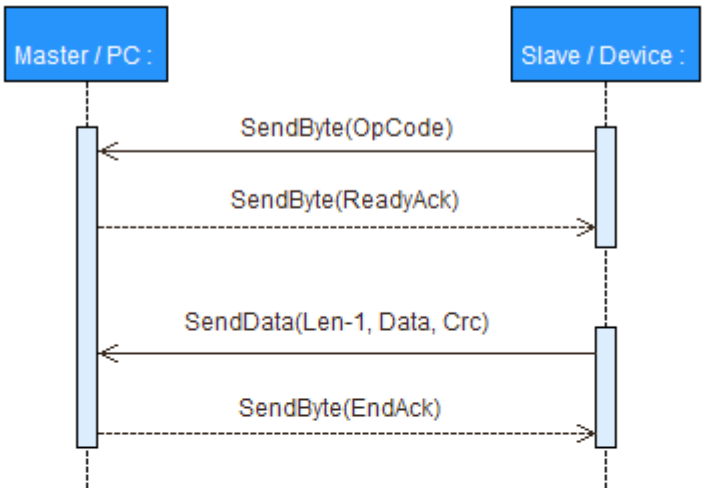


Figure 14: RS232 Communication – Receiving a response data frame from device

6.3 Frame Structure

6.3.1 Overview

The data bytes are sequentially transmitted in frames. A frame composes of:

- a header
- a variably long data field and
- a 16-bit long cyclic redundancy check (CRC) for verification of data integrity

OpCode (8-bit)	Len-1 (8-bit)	Data[0] (16-bit)	...	Data[Len-1] (16-bit)	CRC (16-bit)
Header		Data			*Crc*

Figure 15: RS232 Communication - Frame Structure

6.3.2 Header

The header consists of 2 bytes. The first field (*OpCode*) determines the type of data frame to be sent or received. The next field (*Len-1*) contains the length of the data fields.

- **OpCode** - Operation command to be sent to the slave
- **Len-1** - represents the number of words (16-bit value) in the data fields. It 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

6.3.3 Data

The data field contains the parameters of the message. This data block must contain at least one word. The low byte of the word is transmitted first.

- **Data[i] -** The parameter word of the command. The low byte is transmitted first.

6.3.4 CRC

The 16-bit CRC checksum. The algorithm used is CRC-CCITT. The CRC calculation includes all bytes of the frame. The data bytes must be calculated as a word. First you will need 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.

6.4 Error Control

6.4.1 CRC 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)**

```
uint16_t CalcFieldCRC(uint16_t* pDataArray, uint16_t numberOfWords)
{
    uint16_t shifter, c;
    uint16_t carry;
    uint16_t 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;
}
```

Figure 16: RS232 Communication – CRC-CCIT Calculation

6.5 Transmission Byte Order

The unit of data memory in the device is a word (16-bit value). To send and receive a word (16-bit) over the serial port of the slave device, 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)*.

6.6 Data Format

Data is transmitted in an asynchronous way, thus each data byte is transmitted individually with its own start and stop bit. The format is

1 Start bit, 8 Data bits, No parity, 1 Stop bit (8N1)

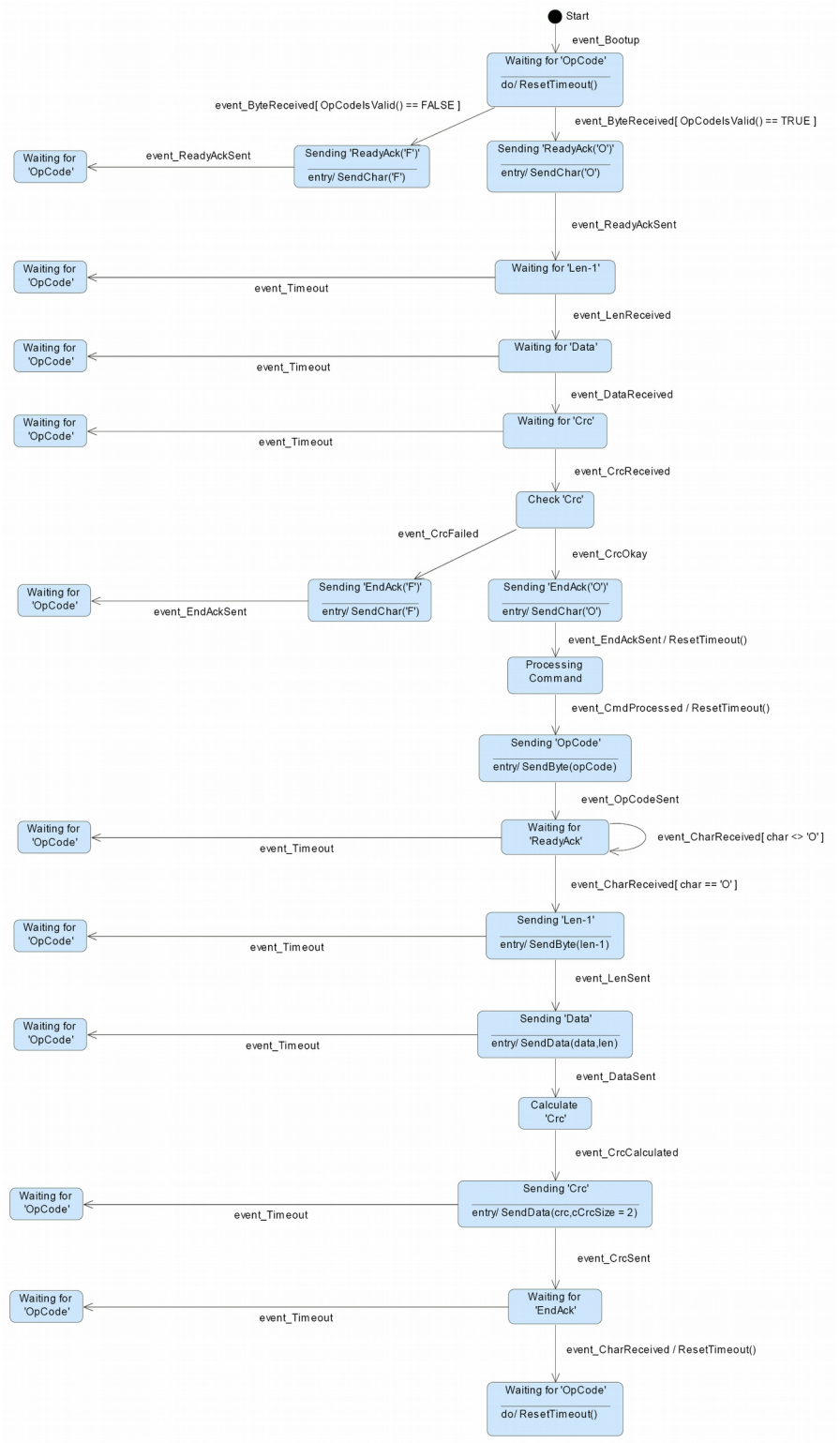
Most serial communication chips (SCI, UART) can generate such data format.

6.7 Timeout Handling

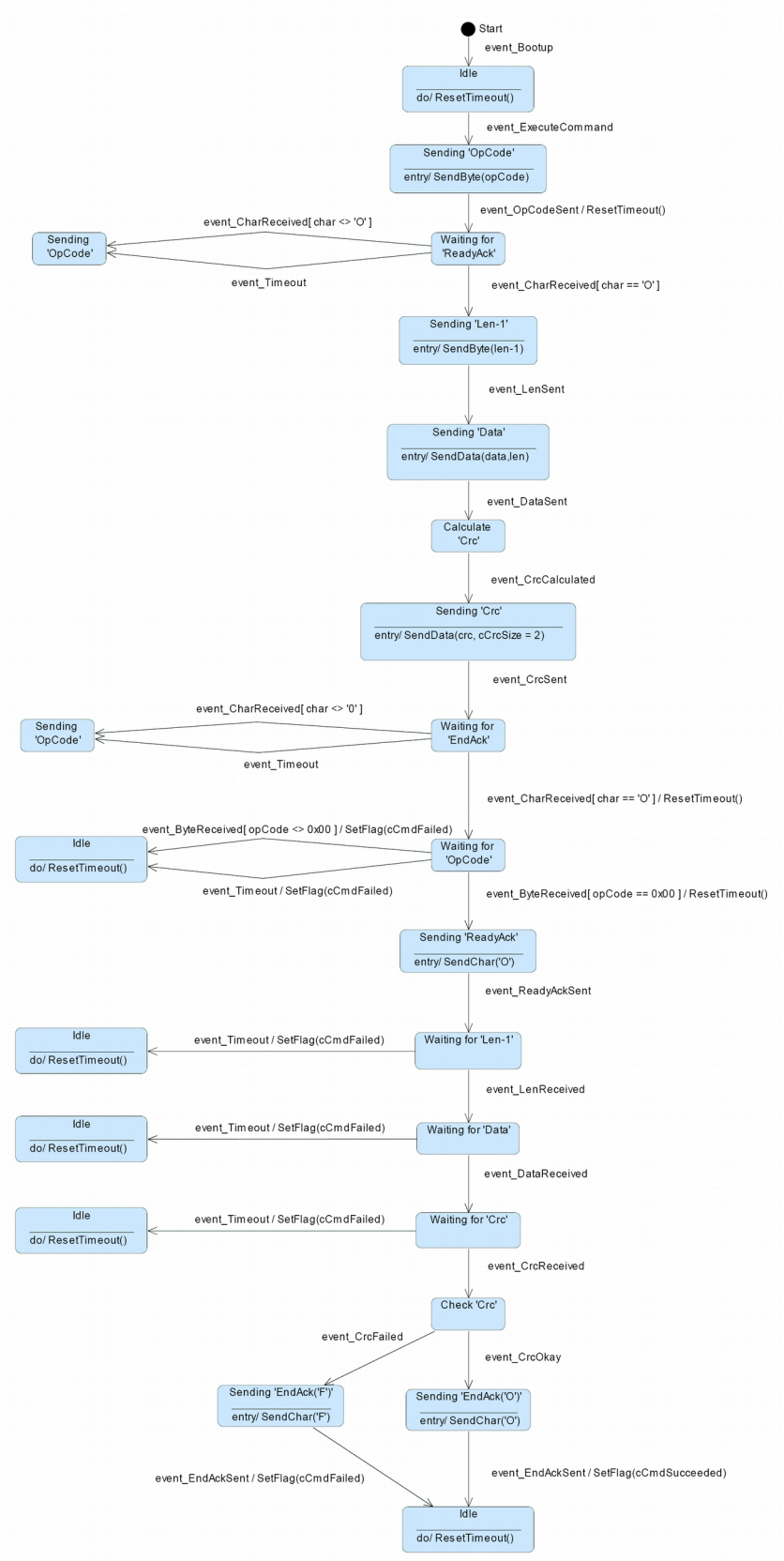
The timeout is handled over a complete frame. Hence, the timeout is evaluated over the sent data frame, the command processing procedure and the response data frame. For each frame (frames, data

processing), the timer is reset and timeout handling will recommence.

6.8 Slave (device) implementation state machine



6.9 Master implementation state machine



6.10 Command Reference

6.10.1 Read Functions

6.10.1.1 READ OBJECT DICTIONARY ENTRY (4 DATA BYTES AND LESS)

Read an object value from the Object Dictionary of the device at the given Index and SubIndex.

READ OBJECT -REQUEST FRAME		
OpCode	0x10	
Len-1	1	
Parameters	WORD Index	Index of Object
	(Low) BYTE SubIndex	SubIndex of Object
	(High) BYTE NodId	Node ID

The device responds with a data frame with 4 bytes of data.

READ OBJECT -RESPONSE FRAME		
OpCode	0x00	
Len-1	3	
Parameters	DWORD ErrorCode	Error Code (see firmware spec.)
	BYTE Data[4]	Data Bytes read

6.10.1.2 READ OBJECT DICTIONARY ENTRY (5 DATA BYTES AND MORE)

INITIATE SEGMENTED READ

Start reading an object value from the Object Dictionary at the given Index and SubIndex with a data size greater than 4 bytes. Because the data does not fit into a single response frame, the transfer is splitted into a number of segments. Use the command *SegmentRead* to read the data.

INITIATE SEGMENTED READ - REQUEST FRAME		
OpCode	0x12	
Len-1	1	
Parameters	WORD Index	Index of Object
	(Low) BYTE SubIndex	SubIndex of Object
	(High) BYTE NodId	Node ID

The device responds with a response frame without any data.

INITIATE SEGMENTED READ - RESPONSE FRAME		
OpCode	0x00	
Len-1	1	
Parameters	DWORD ErrorCode	Error Code (see firmware spec.)

SEGMENT READ

Read a data segment of the object initiated with the command *InitiateSegmentedRead*.

READ SEGMENT - REQUEST FRAME			
OpCode	0x14		
Len-1	0		
Parameters	(Low) Byte ControlByte	[Bit 0...5]	Not used
		[Bit 6]	Toggle Bit
		[Bit 7]	Not used
	(High) BYTE Dummy	Byte without meaning	

The device responds with a data segment of up to 63 bytes of data. The segmented respons frame contains a control byte that indicates the number of data bytes and if there are more segments to read.

READ SEGMENT - RESPONSE FRAME			
OpCode	0x00		
Len-1	2...33		
Parameters	DWORD ErrorCode	Error Code (see firmware spec.)	
	(Low) BYTE ControlByte	[Bit 0...5]	Number of data bytes
		[Bit 6]	Toggle Bit
		[Bit 7]	1 = More segments to read
	BYTE Data[0...63]	Data Bytes read	

6.10.2 Write Functions

6.10.2.1 WRITE OBJECT DICTIONARY ENTRY (4 DATA BYTES AND LESS)

Write an object value to the Object Dictionary at the given Index and SubIndex.

WRITE OBJECT - REQUEST FRAME		
OpCode	0x11	
Len-1	3	
Parameters	WORD Index	Index of Object
	(Low) BYTE SubIndex	SubIndex of Object
	(High) BYTE NodId	Node ID
	BYTE Data[4]	Data Bytes to write

The device responds with a response frame without any data.

WRITE OBJECT - RESPONSE FRAME		
OpCode	0x00	
Len-1	1	
Parameters	DWORD ErrorCode	Error Code (see firmware spec.)

6.10.2.2 WRITE OBJECT DICTIONARY ENTRY (5 DATA BYTES AND MORE)

INITIATE SEGMENTED WRITE

Start writing an object value to the Object Dictionary at the given Index and SubIndex with a data size of more than 5 bytes. The transfer is splitted into segments and initiated with this *InitiateSegmentedWrite* frame. Use the command *SegmentWrite* to write the data.

INITIATE SEGMENTED WRITE - REQUEST FRAME		
OpCode	0x13	
Len-1	3	
Parameters	WORD Index	Index of Object
	(Low) BYTE SubIndex	SubIndex of Object
	(High) BYTE NodId	Node ID
	DWORD ObjectLength	Total number of bytes to write.

The device acknowledges the start of the segmented transfer with the following response frame

INITIATE SEGMENTED WRITE - RESPONSE FRAME		
OpCode	0x00	
Len-1	1	
Parameters	DWORD ErrorCode	Error Code (see firmware spec.)

SEGMENT WRITE

Write a data segment to the object initiated with the command InitiateSegmentedWrite.

WRITE SEGMENT - REQUEST FRAME			
OpCode	0x15		
Len-1	0...31		
Parameters	(Low) Byte ControlByte	[Bit 0...5]	Number of data bytes
		[Bit 6]	Toggle Bit
		[Bit 7]	Not used
	BYTE Data[0...63]	Data bytes to write	

The device acknowledges the segment with the following response frame.

WRITE SEGMENT - RESPONSE FRAME			
OpCode	0x00		
Len-1	2		
Parameters	DWORD ErrorCode	Error Code (see firmware spec.)	
	(Low) BYTE ControlByte	[Bit 0...5]	Number of data bytes
		[Bit 6]	Toggle Bit
		[Bit 7]	Not used
	(High) BYTE Dummy	Byte without meaning	

6.11 Example Frames

6.11.1 Reading Object 0x1000 – Device Type

Index	Sub Index	Name	Type	Access	Value
0x1000	0x00	Device Type	UInt32	RO	0x000200192

The following example shows, how to read the device type object. The device type object can be read via object dictionary index 0x1000 and sub-index 0. In the following example the object is read from a neMESYS pump and the value returned by the device is 0x00020192.

r/w	Data	Comment
w	10	First we write 0x10 to indicate a data read
r	4F	Then we wait for 'O' that indicates the OK. The ASCII value for 'O' is 0x4F
w	01 00 10 00 02 10 CD	Now we send the read request frame with all required data fields: Number of Words without CRC – 1, Object Index 0x1000 (little endian), Sub Index 0, Node ID 2, CRC Checksum
r	4F	Then we wait for 'O' that indicates the OK. The ASCII value for 'O' is 0x4F
r	00	Now we wait for the response. 0x00 indicates the start of the response
w	4F	We acknowledge the reception of the response with OK = 0x4F
r	03 00 00 00 00 92 01 02 00 EB 6D	The device sends a read response frame: Number of Words without CRC – 1, Error Code, Data Bytes read little Endian: 0x00020192, CRC Checksum
w	4F	We calculate the checksum and acknowledge the reception of the response with OK (0x4F) if checksum is ok.

6.11.1.1 CALCULATING THE CRC CHECKSUM FOR THE READ REQUEST FRAME

Before you calculate the CRC checksum for the read request frame, you should have the following array of data words:

DataArray[0]	0x1001	HighByte(OpCode 10 - read) + LowByte(len-1);
DataArray[1]	0x1000	Object Index 0x1000
DataArray[2]	0x0200	Node ID 2, Sub Index 0
DataArray[3]	0x0000	0x0000 – Placeholder for CRC

Now you can calculate the checksum for the 4 words and insert the result 0xCD10 into the DataArray[3] field in little endian order so that you get the following frame: 01 00 10 00 02 10 CD

6.11.1.2 VERIFYING THE CRC CHECKSUM OF THE READ RESPONSE FRAME

If you have received the read response frame 03 00 00 00 00 92 01 02 00 EB 6D, then you should have the following array of data words for CRC calculation:

DataArray[0]	0x0003	HighByte(OpCode 00 - response) + LowByte(len-1);
DataArray[1]	0x0000	Error code low word
DataArray[2]	0x0000	Error code high word
DataArray[3]	0x0192	Data low word
DataArray[4]	0x0002	Data high word
DataArray[5]	0x6DEB	CRC checksum

Now you can calculate the checksum for the 6 words. The result of the CRC calculation should be 0. Only if the result is 0, you have received a valid data frame.

6.11.2 Writing Object 0x1017 – Producer Heartbeat Time

Index	Sub Index	Name	Type	Access	Value
0x1017	0x00	Producer Hearbeat Time	UInt16	RW	1000

The following example shows, how to write a producer heartbeat time of 1000 milliseconds into the object 0x1017 sub-index 0.

r/w	Data	Comment
w	11	First we write 0x11 to indicate a data write
r	4F	Then we wait for 'O' that indicates the OK. The ASCII value for 'O' is 0x4F
w	03 17 10 00 02 E8 03 00 00 10 DC	Now we send the data frame with all required data fields: Number of Words without CRC – 1, Object Index 0x1017 (little endian), Sub Index 0, Node ID 2, the value 1000 (hex 0x3E8) little endian, CRC Checksum
r	4F	Then we wait for 'O' that indicates the OK. The ASCII value for 'O' is 0x4F
r	00	Now we wait for the response. 0x00 indicates the start of the response
w	4F	We acknowledge the reception of the response with OK = 0x4F
r	01 00 00 00 00 51 AA	The device sends a write response frame: Number of Words without CRC – 1, Error Code, CRC Checksum
w	4F	We calculate the checksum and acknowledge the reception of the response with OK (0x4F) if checksum is ok.

6.11.3 Reading Object 0x6041 – Statusword of a nemesys syringe pump

Index	Sub Index	Name	Type	Access	Value
0x6041	0x00	Statusword	UInt16	RO	0x508

The following example shows, how to read the statusword of a nemesys syringe pump. The statusword can be read via object dictionary index 0x6041 and sub-index 0. In the following example the value returned by the device is 0x508. The device has the node ID 2.

r/w	Data	Comment
w	10	First we write 0x10 to indicate a data read
r	4F	Then we wait for 'O' that indicates the OK. The ASCII value for 'O' is 0x4F
w	01 41 60 00 02 F8 A5	Now we send the data frame with all required data fields: Number of Words without CRC – 1, Object Index 0x6041 (little endian), Sub Index 0, Node ID 2, CRC Checksum
r	4F	Then we wait for 'O' that indicates the OK. The ASCII value for 'O' is 0x4F
r	00	Now we wait for the response. 0x00 indicates the start of the response
w	4F	We acknowledge the reception of the response with OK = 0x4F
r	03 00 00 00 00 08 05 00 00 A0 38	The device sends a read response frame: Number of Words without CRC – 1, Error Code, Data Bytes read little Endian: 0x508, CRC Checksum
w	4F	We calculate the checksum and acknowledge the reception of the response with OK (0x4F) if checksum is ok.

6.11.4 Writing Object 0x6040 – Controlword of nemesys syringe pump

Index	Sub Index	Name	Type	Access	Value
0x6040	0x00	Controlword	UInt16	RW	0x80

The following example shows how to write the controlword of a nemesys syringe pump. The controlword can be written via object dictionary index 0x6040 and sub-index 0. In the following example the value written to the device is 0x80. The device has the node ID 2.

r/w	Data	Comment
w	11	First we write 0x11 to indicate a data write
r	4F	Then we wait for 'O' that indicates the OK. The ASCII value for 'O' is 0x4F
w	03 40 60 00 02 80 00 00 00 D9 36	Now we send the data frame with all required data fields: Number of Words without CRC – 1, Object Index 0x6040 (little endian), Sub Index 0, Node ID 2, the value 0x80 little endian, CRC Checksum
r	4F	Then we wait for 'O' that indicates the OK. The ASCII value for 'O' is 0x4F
r	00	Now we wait for the response. 0x00 indicates the start of the response
w	4F	We acknowledge the reception of the response with OK = 0x4F
r	01 00 00 00 00 51 AA	The device sends a write response frame: Number of Words without CRC – 1, Error Code, CRC Checksum
w	4F	We calculate the checksum and acknowledge the reception of the response with OK (0x4F) if checksum is ok.

6.11.5 Writing Object 0x607A – Target Position of nemesys syringe pump

Index	Sub Index	Name	Type	Access	Value
0x607A	0x00	Target Position	Int32	RW	0x80

The following example shows how to write the target position of a nemesys syringe pump. The target position can be written via object dictionary index 0x607A and sub-index 0. In the following example the value written to the device is 280000 (0x445C0). The device has the node ID 2.

r/w	Data	Comment
w	11	First we write 0x11 to indicate a data write
r	4F	Then we wait for 'O' that indicates the OK. The ASCII value for 'O' is 0x4F
w	03 7A 60 00 02 C0 45 04 00 EA 13	Now we send the data frame with all required data fields: Number of Words without CRC – 1, Object Index 0x607A (little endian), Sub Index 0, Node ID 2, the value 280000 (0x445C0) little endian, CRC Checksum
r	4F	Then we wait for 'O' that indicates the OK. The ASCII value for 'O' is 0x4F
r	00	Now we wait for the response. 0x00 indicates the start of the response
w	4F	We acknowledge the reception of the response with OK = 0x4F
r	01 00 00 00 00 51 AA	The device sends a write response frame: Number of Words without CRC – 1, Error Code, CRC Checksum
w	4F	We calculate the checksum and acknowledge the reception of the response with OK (0x4F) if checksum is ok.

7 Pump Control

7.1 Drive Control Overview

Internally the pump uses a EPOS CANopen DS402 servo drive to move the pusher and the syringe piston. A detailed description of the EPOS CANopen drives is provided with the document EPOS2-Firmware-Specification-En.pdf. You can control the drive by reading and writing the object dictionary entries of the device. The controller has an extensive object directory (see section *8 Object Dictionary* in the *EPOS firmware specification*), but only a few entries are relevant for the control of the neMESYS pump. The following table list all object dictionary entries that are required for pump control.

Index	Name	Data Type	Access
0x1001	Error Register	UNSIGNED8	RO
0x1003	Error History (Predefined Error Field)	ARRAY	RO
0x2003	RS232 Frame Timeout	UNSIGNED16	RW
0x200C	Custom persistent memory	RECORD	RW
0x2028	Velocity Actual Value Averaged	INTEGER32	RO
0x2078	Digital Output Functionalities	RECORD	RW
0x207C	Analog Inputs	ARRAY	RO
0x2081	Home Position	UNSIGNED32	RW
0x2210	Position Sensor Configuration	RECORD	RW
0x6040	Control Word	UNSIGNED16	RW
0x6041	Status Word	UNSIGNED16	RO
0x6060	Modes of Operation	INTEGER8	RW
0x6061	Modes of Operation Display	INTEGER8	RO
0x6064	Position Actual Value	INTEGER32	RO
0x607A	Target Position	INTEGER32	RW
0x607C	Home Offset	INTEGER32	RW
0x6081	Profile Velocity	UNSIGNED32	RW
0x608B	Velocity Notation Index	INTEGER8	RW
0x6098	Homing Method	INTEGER8	RW
0x6099	Homing Speeds	ARRAY	RW

7.2 Operating Modes

The EPOS CANopen drive supports a number of operating modes (see section *5 Operating Modes* in the EPOS2-Firmware-Specification-En.pdf). For pump control, only two of these operation modes are required.

- **MODE 6 – HOMING MODE:** This mode is required for reference move of the pusher and to initialize the internal position monitioring
- **MODE 1 – PROFILE POSITION:** this mode is required for normal pumps tasks like aspirating or dispensing

To activate a mode, you simply need to write the mode index (e.g. 1 for Profile Position) into the object dictionary entry *0x6060 Modes Of Operation*. To read out the active operation mode, you simple need to read the current value of *0x6061 Modes Of Operation Display*.

Index	Subindex	Object	Description
0x6060	0	Modes of Operation	Write into this object to switch operating mode
0x6061	0	Modes of Operation Display	Read current operating mode from this object

7.3 Translation of volume / flow units

7.3.1 Introduction

Whenever you call a function that requires a volume (position) value or a flow (speed) value, the value is given in device internal units like increments for position values and mrpm (millirevolutions per minute) for velocity values. These technical units are not well suited for dosing tasks (for volumes and flow rates) and need to be translated by the application to implement pump control.

This translation depends on several parameters like mechanical configuration of the single dosing units (gear) and it also depends on the syringes used for dosing. The following two sections will show you, how you can convert the internal device units for device control into units for volume and flow values.

7.3.2 Reading out device parameters

7.3.2.1 OVERVIEW

For the calculation of flow rates and volumes the following device parameters are required:

Index	Subindex	Object	Description
0x608B	0	Velocity Notation Index	Read velocity notation from this object
0x2210	0	Encoder Resolution	Read the encoder resolution from this object
0x200C	1	Custom Persistent memory 1	Gear Factor Numerator
0x200C	4	Custom Persistent memory 4	Gear Factor Denominator

7.3.2.2 VELOCITY NOTATION INDEX

The velocity notation index defines the unit prefix that precedes the velocity base unit rpm (revolutions per meter) and thus defines the internal velocity unit of the drive device. To get the velocity notation index you need to read the object dictionary entry *0x608B Velocity Notation Index*. All velocity values are given and returned in internal velocity units. The following values are possible:

Value	Velocity unit
0	rpm (revolutions per minute)
-1	drpm (decirevolutions per minute)
-2	crpm (centirevolutions per minute)
-3	mrpm (millirevolutions per minute)

The default unit is mrpm – millirevolutions per minute.

7.3.2.3 ENCODER RESOLUTION

The encoder resolution defines the number of pulses per motor revolution – increments per motor revolution. All internal position values are given in increments. To read out the encoder pulse number you need to read the object dictionary entry *0x2210, Subindex 1 – Pulse Number Encoder 1*. Because the encoder returns a quadrature signal, the encoder pulse number needs to get multiplied with 4 to get the encoder resolution:

Encoder Resolution = Pulse Number Encoder x 4

If the Pulse Number of the encoder is 512, then the resulting encoder resolution is 512 x 4 = 2048.

7.3.2.4 GEAR FACTOR

The gear factor defines the factor for the conversion of motor revolutions into the moved pusher distance in mm. The gear factor consists of a gear nominator and a gear denominator. You need to read the following two object dictionary entries, to get the gear factor.

Parameter	Object	Subindex
Gear Factor Numerator	0x200C Custom Persistent memory	1
Gear Factor Denominator	0x200C Custom Persistent memory	4

From these to values, you can calculate the gear factor:

Gear Factor (rev/mm)= Gear Nominator / Gear Denominator.

7.3.3 Position value conversion

7.3.3.1 CALCULATING THE POSITION CONVERSION FACTOR

With the values read from the device, you can calculate a position conversion factor for conversion between internal device units (increments) and millimetres. First we can convert the increment value into motor revolutions with the help of the encoder resolution value:

Motor revolutions =Increments / Encoder Resolution

Then you have to translate these motor rotations into the distance of the pusher with the help of the gear factor:

Distance in mm = Motor revolutions / Gear factor

So the final calculation is:

$$mm = \frac{Increments}{Encoder\ Resolution(inc/rev) \times Gear\ factor(rev/mm)}$$

From this formula we can extract the position conversion factor:

Position conversion factor (inc/mm) = Encoder Resolution (inc/rev) x Gear Factor (rev/mm)

7.3.3.2 CONVERSION OF POSITION VALUES

To convert from increments into a distance in mm, you just need to de divide the increments value by the position conversion factor:

mm = Increments / Position conversion factor

To convert a distance in mm into an increments value, you just need to multiply the distance with the conversion factor:

Increments = mm * Position conversion factor

7.3.3.3 EXAMPLE POSITION CONVERSION

The following example shows how to convert a distance in millimetres into internal position units:.

Encoder resolution:	2048 inc/rev
Gear factor:	14,0625 rev/mm
Position conversion factor	2048 inc/rev * 14,0625 rev/mm = 28.800 inc/mm
Distance in mm:	10 mm
Position value in increments:	10 mm * 28.800 inc/mm = 288.000 inc

7.3.4 Velocity value conversion

7.3.4.1 CALCULATING THE VELOCITY CONVERSION FACTOR

With the values read from the device, you can calculate a velocity conversion factor for conversion between internal device units and millimetres/second (mm/s). First we convert the internal velocity unit into into revolutions/minute.

$$rev/min = device\ velocity \times 10^{velocity\ notation\ index}$$

Then we can convert the revolutions/minute into revolutions per second by dividing by 60.

$$rev/s = rev/min\ /\ 60\ s/min$$

Finally we can calculate the velocity in millimetres/second with the help of the gear factor:

$$mm/s = \frac{rev/s}{Gear\ factor\ (rev/mm)}$$

So the final calculations is:

$$mm/s = \frac{device\ velocity \times 10^{velocity\ notation\ index}}{60\,s/min \times Gear\ factor\ (rev/mm)}$$

From this formula we can extract the velocity conversion factor:

$$Velocity\ conversion\ factor = \frac{60\,s/min \times Gear\ factor\ (rev/mm)}{10^{velocity\ notation\ index}}$$

7.3.4.2 CONVERSION OF VELOCITY VALUES

To convert from device velocity into a velocity in mm/s, you just need to divide device velocity values by the velocity conversion factor:

mm/s = Device velocity / Velocity conversion factor

To convert a velocity in mm/s into a device velocity value, you just need to multiply the velocity with the conversion factor:

Device velocity = mm/s x Velocity conversion factor

7.3.4.3 EXAMPLE VELOCITY CONVERSIONS

Velocity notation index:	-3 (millirevolutions per minute)
Gear factor:	14,0625 rev/mm
Velocity conversion factor:	60s/min x 14,0625 rev/mm / 10^{-3} = 843.750
Velocity value:	2mm/s
Device velocity:	2 mm/s * 843.750 = 1.687.500 mrev/min

7.3.5 Volume value conversions

7.3.5.1 CALCULATION

Section Position value conversion shows, how to convert internal device position into millimetres. This section shows, how to convert a position value in millimetres into a volume in millilitres. To convert a pusher movement in millimetres into a volume value in millilitres, you need to know the inner diameter of the syringe mounted on the device. With the help of the inner syringe diameter and a length in millimetres, you can calculate the cylinder volume in mm^3.

$$Volume(mm^3) = \frac{\pi}{4} d(mm)^2 \cdot length(mm)$$

One millilitre is equal to 1000 mm^3. So you can calculate millilitres directly with the following formula:

$$Volume(ml) = \frac{\pi}{4} d(mm)^2 \cdot length(mm) / 1000$$

From a given value in millilitres you can calculate the pusher distance with the following formula:

$$mm = \frac{Volume(ml) \cdot 1000 \cdot 4}{\pi d^2}$$

With then help of the Position conversion factor you can now convert millimetres into internal device position units (increments).

7.3.5.2 EXAMPLE VOLUME CONVERSION

The following example shows, how to convert a volume value in millilitres into internal device position units:

Volume:	10 ml
Inner syringe diameter:	14,5673 mm
Distance in mm:	10 ml * 1000 mm^3/ml * 4 / π / (14,5673 mm)^2 = 60 mm
Position conversion factor:	28.800 inc/mm
Position value in increments:	60 mm * 28.800 inc/mm = 1.728.000 inc

7.3.6 Flow value conversions

7.3.6.1 CALCULATION

Section Velocity value conversion shows, how to convert internal device velocity into millimetres/second (mm/s) and vice versa. This section shows, how to convert a velocity value in millimetres/second (mm/s) into a flow value in millilitres/second (ml/s). To convert a pusher movement in millimetres/second into a flow value in millilitres/second, you need to know the inner diameter of the syringe mounted on the device. With the help of the inner syringe diameter and a length in millimetres, you can calculate the cylinder volume in mm^3.

$$Volume(mm^3) = \frac{\pi}{4} d(mm)^2 \cdot length(mm)$$

One millilitre is equal to 1000 mm^3. So you can calculate millilitres directly with the following formula:

$$Volume(ml) = \frac{\pi}{4} d(mm)^2 \cdot length(mm)/1000$$

Now we can easily create the formula for conversion of velocity values in mm/s into flow values in ml/s

$$Flow(ml/s) = \frac{Volume(ml)}{s} = \frac{\pi d(mm)^2}{4 \times 1000} \times Velocity(mm/s)$$

and the formula for conversion of flow values in ml/s into velocity values in mm/s

$$Velocity(mm/s) = \frac{Flow(ml/s) \cdot 1000 \cdot 4}{\pi d(mm)^2}$$

With the help of the Velocity conversion factor you can now convert mm/s into internal device velocity.

7.3.6.2 EXAMPLE FLOW CONVERSION

The following example shows, how to convert a flow value in millilitres/second (ml/s) into internal device velocity units:

Flow:	1,054814 ml/s
Inner syringe diameter:	14,5673 mm
Velocity in mm/s:	1,054814 ml/s * 1000 mm3/ml * 4 / π / (14,5673 mm)2 = 6,328 mm/s
Velocity conversion factor:	843.750
Device velocity:	6,328 mm/s * 843.750 = 5.339.250 mrev/min

7.4 Enabling drive

7.4.1.1 EXAMPLE FLOW CONVERSION

Before you can move the pusher, you need to set the pump drive into Operation Enabled state. Operation Enabled means, the drive function is enabled and power is applied to the motor. Right after power on or after a reset, the drive is not in Operation Enabled state. To set the drive into Operation Enabled state, you need to control the internal drive state machine via the objects *0x6040 Controlword* and *0x6041 Statusword*.

For a detailed description how to control the drive state machine and how to set the drive into Operation Enabled state, please read the section *3.2 Device Control* in the EPOS2-Firmware-Specification-En.pdf document.

Index	Subindex	Object	Description
0x6040	0	Controlword	Write to this object to control the internal device state machine
0x6041	0	Statusword	Read current device state and status information from this object

7.5 Initializing position counter (Homing)

7.5.1 Overview

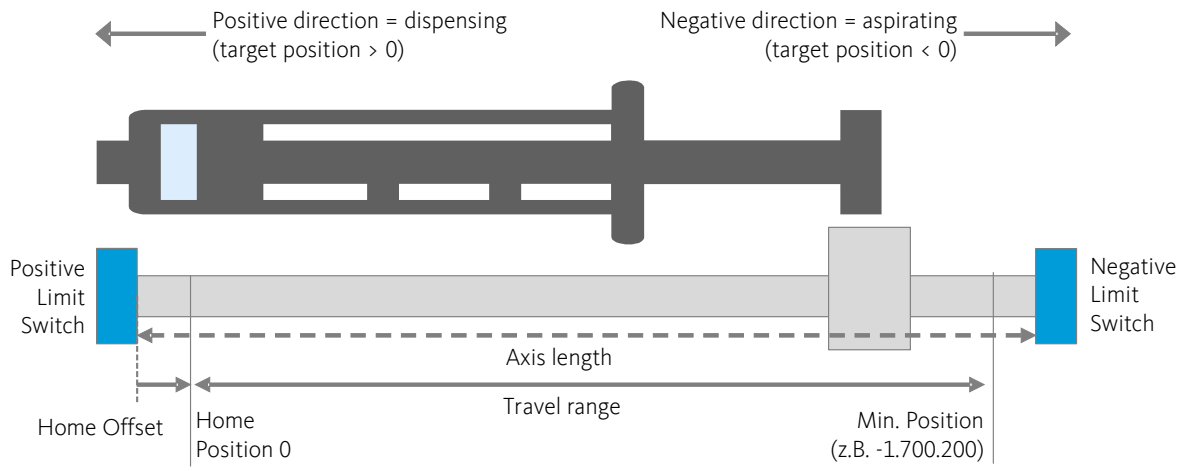
The neMESYS drive units use a relative encoder for position tracking and supervision. “Relative” means, that all position information will be lost, if drive is powered of. Right after power on, the internal position counter is always zero no matter where the pusher is located at the moment. To properly initialize the internal position counter you must either perform a homing move, or load a previously saved position value. The following two sections will show you, how to do this.

7.5.2 Homing move

7.5.2.1 ACTIVATING HOMING MODE

To initialize the internal position counter, you can execute a homing move. That means, the pusher will move to a known position to initialize its internal counter. A known position is one of the two limit switches:

- the positive limit switch – the position where the syringe is completely empty
- and the negative limit switch – the position where the is completely filled



Before you can start a homing move, you need to activate the homing mode by writing the value *6* into object *0x6060 Modes of Operation* – see section Operating Modes.

The following objects are required for homing mode:

Index	Subindex	Object	Description
0x6098	0	Homing Method	
0x6099	1	Speed for switch search	The homing speed for searching the limit switch
0x6099	2	Speed for zero search	The homing speed for searching the zero position
0x607C	1	Home Offset	The distance the device moves away from the limit switch

For a detailed description of the homing mode, read the section 5.*4 Homing Mode* in the EPOS2-Firmware-Specification-En.pdf document.

7.5.2.2 SETTING HOMING METHOD

The drive supports the following two homing methods when executing a homing move:

Value	Method
1	Negative Limit Switch
2	Positive Limit Switch

The default homing method is *2 – Positive Limit Switch*. The positive limit switch is the limit switch where the syringe is completely empty. To set the homing method, simply write the homing method into object *0x6098 Homing Method*.

When the drive performs a homing move, it does the following steps:

- the initial direction of the movement is to the limit switch if the limit switch is inactive
- the axis moves with speed for switch search (*0x6099 Speed for switch search*) into the limit switch – until the switch becomes active
- now the axis moves witch speed for zero search (0x6099 Speed for zero search) away from the limit switch, to the edge of the switch until it becomes inactive
- Now, the axis moves the *0x607C Home Offset*. This point will be used as reference for all further moves and is set to Home Position.

HINT. The default homing method used in neMESYS UserInterface software is method *2 – Positive Limit Switch.*

7.5.2.3 SETTING HOMING SPEEDS

You can set the homing speed by writing to the following two object dictionary entries.

Index	Subindex	Object	Description
0x6099	1	Speed for switch search	The homing speed for searching the limit switch
0x6099	2	Speed for zero search	The homing speed for searching the zero position

HINT. You do not need to change these speeds. If you do not write to these entries, the homing move will be performed with the default homing speeds.

7.5.2.4 SETTING HOME OFFSET

The home offset is the distance from the limit switch edge to the real zero position of the device. That

means, the device always moves the home offset from the limit switch edge. The home offset ensures, that there will be a small distance between the zero position and the limit switch. This ensures, that the drive does not activate the limit switch, when it moves to the zero. To change the home offset, write to the following object dictionary entry:

Index	Subindex	Object	Description
0x607C	0	Home Offset	The distance the device moves away from the limit switch

HINT. You do not need to change the home offset. If you do not write to this entry, the homing move will be performed with the default homing offset configured by cetoni.

7.5.2.5 STARTING HOMING MOVE

You can start the homing move by writing to the Controlword. For a detailed description of the homing mode specific Controlword and Statusword, please read the section 5.*4 Homing Mode* in the EPOS2-Firmware-Specification-En.pdf document.

7.5.3 Restoring position counter

7.5.3.1 OVERVIEW

To avoid a homing move each time you turn on your pump device, you can restore a previously saved position counter value. The following sections show you, how to save the position counter and how to restore it.

7.5.3.2 SAVE POSITION COUNTER

To save the position counter, you just need to read the *0x6064 Position Actual Value* object, and store its value persistently into a file or any other persistent memory. You need to do this, each time before you turn off your device.

Index	Subindex	Object	Description
0x6064	0	Position Actual Value	The current position value in increments

7.5.3.3 RESTORING POSITION COUNTER

To restore the position counter you need to perform the following steps.

(1) First you need to activate the homing mode, by writing the value *6 (homing mode)* into object *0x6060 Modes of Operation* (see section Activating Homing Mode).

(2) Then you need to set the homing method *35 (Actual Position)* by writing into object *0x6098 Homing Method*.

(3) Then you can load you previously saved position counter value from file and write it into object *0x2081 Home Position*.

(4) Now you need to enable the drive by writing to the object *0x6040 Controlword* (see section Enabling drive)

(5) Finally you can start the homing by setting the Homing Start bit in object *0x6040 Controlword* (see 5.4 *Homing Mode* in the EPOS2-Firmware-Specification-En.pdf document)

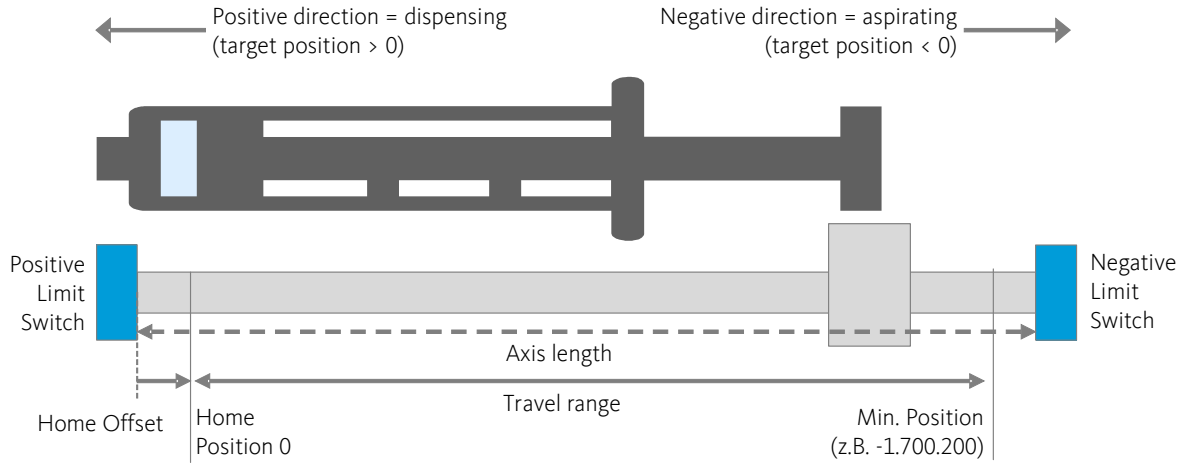
The following object are used for restoring the position counter value:

Index	Subindex	Object	Description
0x6098	0	Homing Method	Method 35 – Actual Position
0x2081	0	Home Position	The position counter value to restore
0x6040	0	Controlword	To start the homing

7.6 Dosing

7.6.1 Introduction

Normally all dosing tasks are performed in *Profile Position Mode*. That means for each dosing task you need to set the volume, the flow rate and you need to start/stop the pump. The following picture shows the device hardware:



The physical *axis length* is the distance between the positive and negative limit switch. During normal dosing operations, the pusher should not reach the limit switches. Therefore the real *travel range* is limited by two offsets: the *Home Offset* on the *Positive Limit Switch* side and a second offset on the *Negative Limit Switch* side.

7.6.2 Reading device configuration

7.6.2.1 OVERVIEW

The object *0x200C Custom Persistent Memory* contains additional information about the device configuration.

Index	Subindex	Object	Description
0x200C	2	Custom Persistent memory 2	Axis length (distance between negative and positive limit switch)
0x200C	3	Custom Persistent memory 3	Axis configuration (pump type, valve...)

7.6.2.2 CALCULATING THE TRAVEL RANGE

To calculate the travel range, you first need to read object *0x200C Subindex 2 Axis Length* to get the

physical axis length between positive and negative limit switch. Now you can read the object *0x607C Home Offset* to get the offset from the zero position to the positive limit switch. Because the offset is valid for both limit switches, you can calculate the travel range with the following formula:

Travel range = Axis Length – 2 x Home Offset

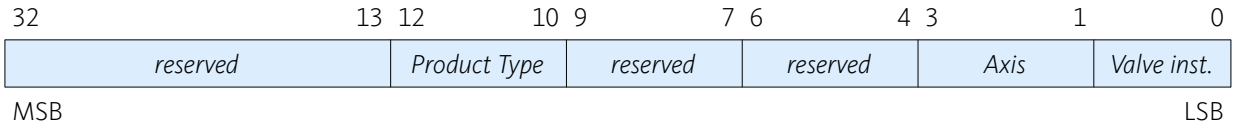
Now you can calculate the minimum and maximum position values, to limit the movement of the pusher. The position counter value increases if the drive moves in positive direction towards the positive limit switch and decreases if it moves in negative direction towards the negative limit switch. Therefore the two position limits are:

Maximum position = 0

Minimum position =0 – Travel range

7.6.2.3 READING AXIS CONFIGURATION

To get additional information about the axis configuration, you can read object *0x200C Subindex 3 Axis Configuration*. The object contains a bitfield with additional information about various axis configuration parameters.



Field	Value	Description
Product Type	0	Low Pressure Module
	1	High Pressure Module
	2	Mid Pressure Module
	3	XL Module
	4	2XL Module
Axis	0	Single Moduel
	1	Left axis of Double Module
	2	Right axis of Double Module
	3	Starter Module
Valve inst.	0	No valve installed
	1	Valve installed

7.6.3 Starting dosage

To start a dosage you need to access the following objects:

Index	Subindex	Object	Description
0x6040	0	Controlword	Write to this object to start / stop dosage
0x6041	0	Statusword	Read current device state and status information from this object
0x607A	0	Target Position	Defines the volume for the next dosage
0x6081	0	Profile Velocity	Defines the flow rate for the next dosage

The pump supports absolute and relative dosing. A relative movement aspirates or delivers a certain volume. The position value that you write into object *0x607A Target Position* is relative to the current position. Write a negative position value to aspirate and a positive position value to displense a certain amount of fluid. After the movement, the syringe content is increased respectively decreased, by the volume, just as a bank account has its balance increased or decreased by a credit or a debit. There is a fixed relationship between the position of the piston and the content in the syringe.

An absolute movement moves the piston of the syringe so that the syringe content reaches the specified value. The position value that you write into object *0x607A Target Position* is an absolute value in the travel range from *Minimum Position – Maximum Position*. The actual movement is a delivering or an aspiration as required to fulfil this purpose, or even no movement at all if the content is already equal to the specified volume.

To start dosage, you need to perform the following steps:

(1) Enable the drive if it is not enabled yet according to the description in section Enabling drive.

(2) Convert the volume and flow values into position and velocity values according to section Translation of volume / flow units.

(3) Write the position value into object *0x607A Target Position*.

(4) Write the velocity value into object *0x6081 Profile Velocity*.

(5) Start the dosage by writing to the object *0x6040 Controlword*. Set the Abs/rel bit in the Controlword to select absolute or relative dosing.

Read the section *5.3 Profile Position Mode* in EPOS2-Firmware-Specification-En.pdf document for a detailed description of the Controlword bits and how to start / stop positioning.

HINT. You only need to write Target Position and Profile Velocity objects, if you want to change the values. If you would like to perform multiple dosing tasks with the same volume or the same flow rate, then you only need to trigger the Controlword.

7.6.4 Stopping dosage

To stop a running dosage set the *Halt* bit in the *Controlword*.

7.7 Valve Switching

If there is a valve mounted on the device or if there is an external valve connected to the neMESYS I/O interface, you can easily switch the valve by writing to the digital outputs.

Index	Subindex	Object	Description
0x2078	1	Digital Output States	Read / write the state of all digital outputs

The valve is connected to the digital outputs *General Purpose C* and *General Purpose D*.

Bit	Nemesys I/O Interface	EPOS Output Name
13	Digital output 1	General Purpose Out C
12	Digital output 2	General Purpose Out D
15	Digital output 3	General Purpose Out A

7.7.1 Switching internal Valve (Nemesys Low Pressure only)

The valves mounted on the pumps are 2/3-way digital valves. These valves have two positions: an *OFF* position (coil power off) and an *ON* position (coil power on). To avoid warming of the valve, and of the fluid that flows through the valve, the valve circuit supports a third state, in which the current coil voltage is lowered. The following logic table shows the valve states and the I/Os.

General Purpose C	General Purpose D	Valve State	Valve LED
0	0	off	off
0	1	on (voltage lowered)	on (dimmed)
1	0	off	off
1	1	on	on

To switch the valve into off state, simply set both outputs to 0. To switch the valve to on state, set both outputs to 1 to get the maximum switching voltage. After one second set *General Purpose C* to 0 to lower the coil voltage and to prevent coil and valve warming.

7.7.2 Switching external Valve connected to I/O interface

The different valves have a different number of switching positions. Depending on the valve type, you need to set the following digital outputs:

Valve Type	Valve Position	Dig. Out 1	Dig. Out 2	Dig. Out 3	Bitmask (Bits 12, 13 and 15)
3-2 Way Valve (first Valve)	Port 1	-	0	-	0x00000000
	Port 2	-	1	-	0x00001000
3-2 Way Valve (second Valve)	Port 1	-	-	0	0x00000000
	Port 2	-	-	1	0x80000000
3-4 Way Contiflow Valve	Closed	0	0	-	0x00000000
	Port 1	1	0	-	0x00001000
	Port 2	0	1	-	0x00002000
	Both ports open	1	1	-	0x00003000
3-3 Way Contiflow Ball Valve	Closed	0	0	-	0x00000000
	Port 1	1	0	-	0x00001000
	Port 2	0	1	-	0x00002000

IMPORTANT. If you write the digital outputs, ensure that you only modify the bits 12, 13 and 15 that are relevant for valve switching. That means you either need to read the value before you change bits or you need to keep an internal shadow register.

7.8 Reading Pressure Sensor / Analog Inputs

The nemesys syringe pumps have two analog inputs. Pressure sensors are normally connected to the first analog input. To read the analog input values you only have to read the two relevant object directory entries:

Index	Subindex	Object	Description
0x207C	1	Analog Input 1	The voltage measured at analog input 1 [mV].
0x207C	2	Analog Input 2	The voltage measured at analog input 2 [mV].

The returned value is the measured voltage in mV in the range from 0 – 5000 mV. If there is a sensor connected to one of the inputs, you just need to translate the voltage value into the sensor value.

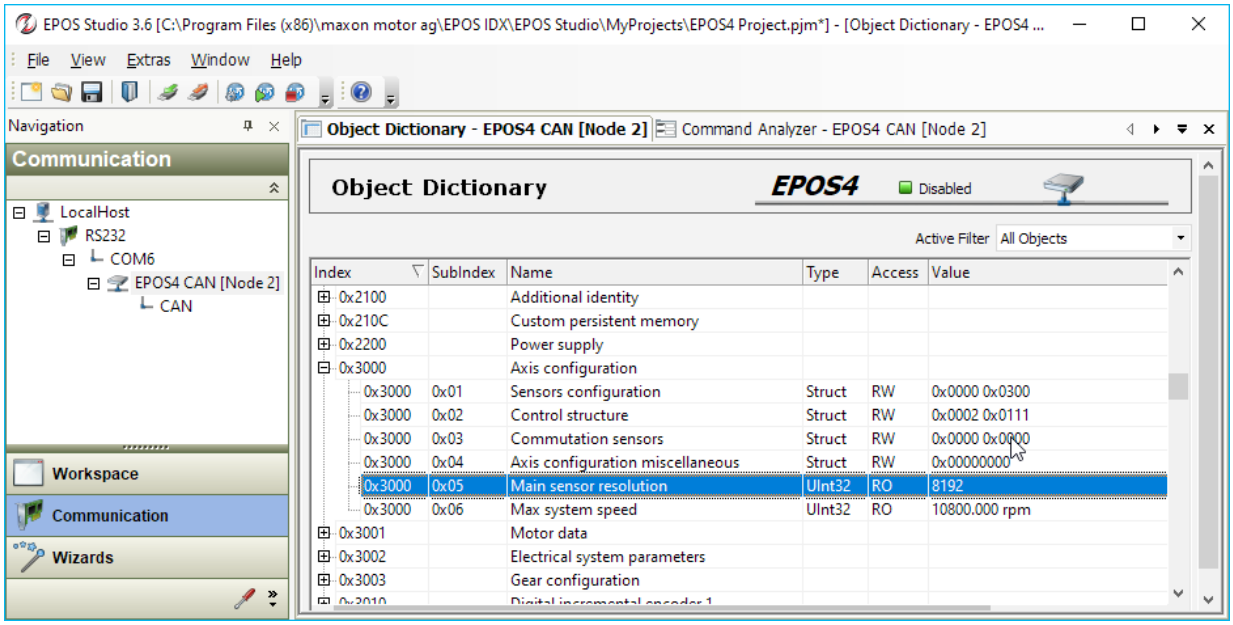
8 Development Tools

8.1 Tools for RS232 Protocol Implementation

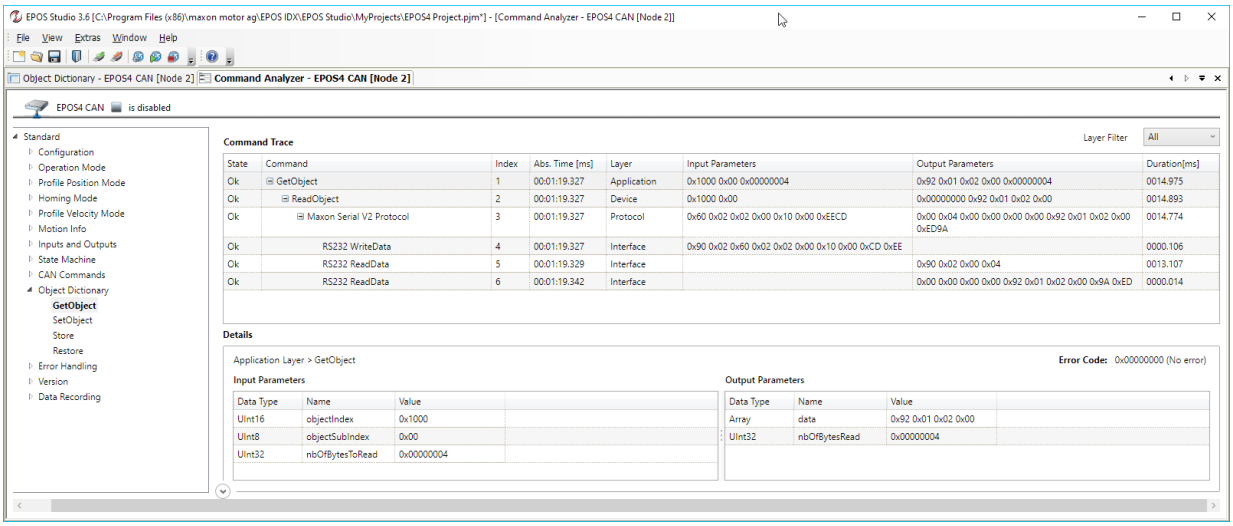
Implementing the RS232 industrial protocol with CRC checksum is somewhat more difficult than implementing a simple ASCII protocol. To simplify and speed up the implementation, find errors in the protocol implementation or to monitor the serial frames, we recommend the following tools:

8.1.1 EPOS Studio

The EPOS Studio software is a powerful tool for access to all device parameters of the pump drive via RS232 or CAN interface. With the EPOS Studio Object Dictionary Tool it is possible to read and write entries of the CANopen object dictionary. With this tool you can modify parameters or verify, if your implementation has properly read or written certain parameters:



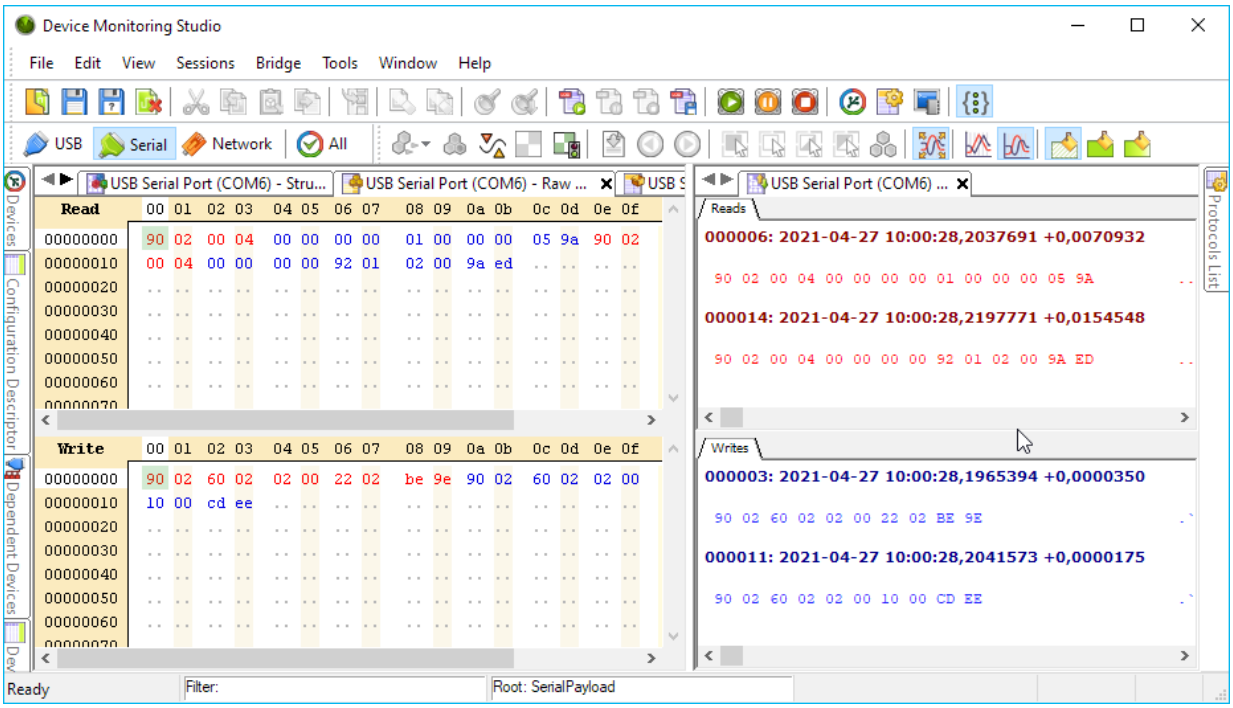
The EPOS Studio Command Analyzer will help you to analyze the low level RS232 protocol including checksum calculation. With this tool you can execute certain commands and access the object dictionary and you will see the corresponding serial protocol frames including CRC and stuff bytes.



You can download the EPOS Studio software here: Download.

8.1.2 Serial Port Monitor

With a serial port monitor, you can monitor the low level data frames on the serial line.



This helps you to see and understand the RS232 frame structure and RS232 checksum calculation. It will also help you, to find and trace errors in your serial protocol implementation. At CETONI we use this serial port monitor: https://www.hhdsoftware.com/serial-port-monitor.

8.1.3 Nemesys V4 RS232 Library Documentation

The Nemesys RS232 Library is an open source implementation of the industrial RS232 protocol in plain C language. The library is well structured and well documented. If you understand C language a little bit, then this library will be a valuable helper for you and you can use it as a template for your implementation. You can browse the online documentation here.

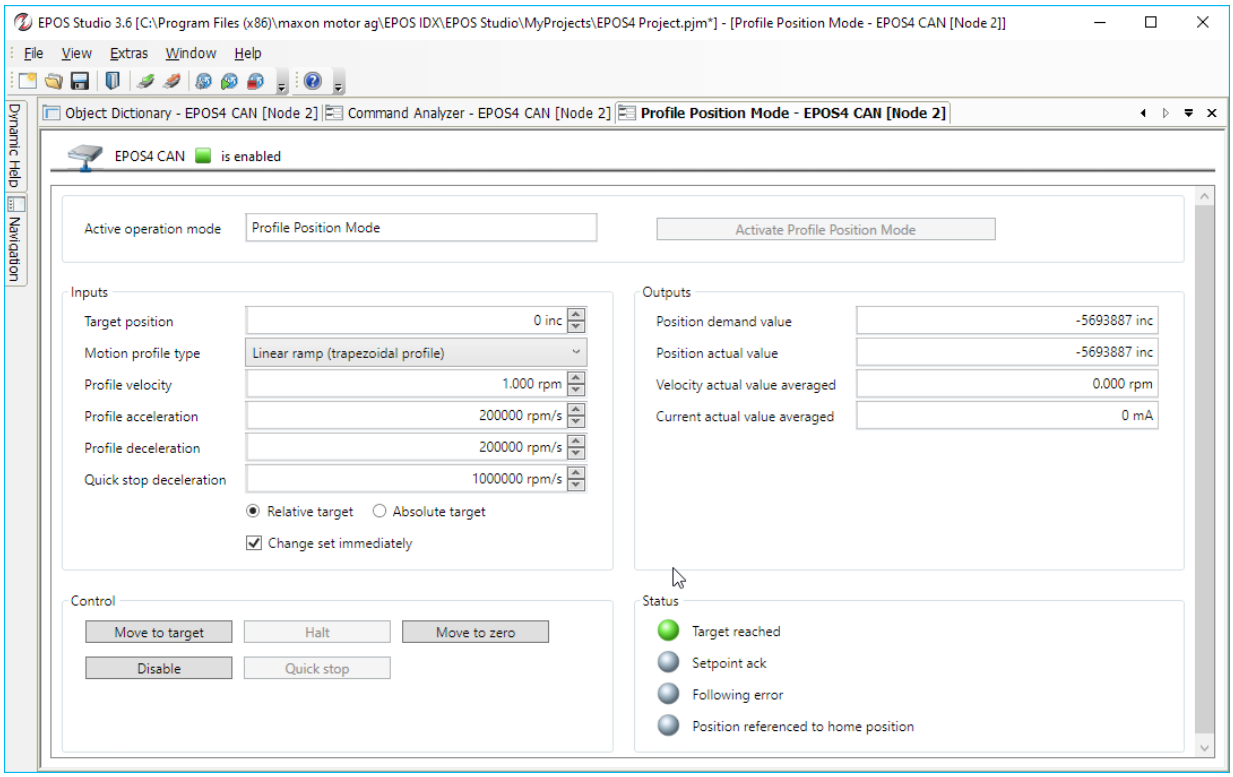
If you would like to learn about the low level serial serial protocol implementation, then you should look into the CSI library. If you would like to learn about the implementation of the Nemesys pump functionality, then you should look into the Nemesys V1 API of the Nemesys RS232 Library.

8.2 Tools for CANopen implementation

8.2.1 EPOS Studio

The EPOS Studio software is a powerful tool for access to all device parameters of the pump drive via RS232 or CAN interface. With the EPOS Studio Object Dictionary Tool it is possible to read and write entries of the CANopen object dictionary. With this tool you can modify parameters or verify, if your implementation has properly read or written certain parameters.

The software also allows you to execute positioning commands via its graphical interface.

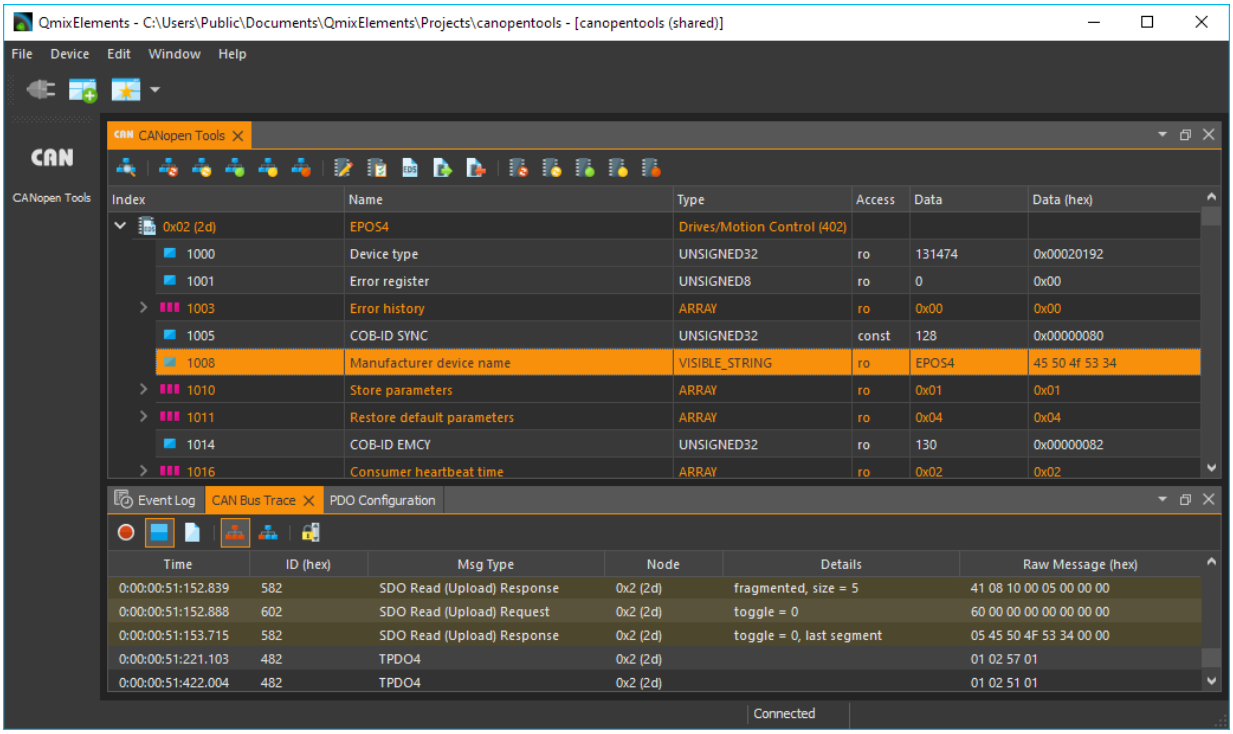


You can download the EPOS Studio software here: Download.

8.2.2 CETONI Elements CANopen Tools Plugin

The CETONI Elements software from CETONI has an CANopen-Tools Plugin which transforms the software into a powerful tool to configure CANopen devices, access the CANopen object dictionary of the Nemesys pumps and to monitor, log and analyze the CAN-Bus traffic and the CANopen protocol of the pumps.

This tool will help you to read and write object dictionary entries and to monitor the CAN-bus traffic of your PLC, PC or embedded control device connected to the Nemesys pumps.



Read the section *CANopen Tools Workbench* in the CETONI Elements manual to learn how to open and use this tool.