EPOS2 P

Programmable Positioning Controllers

Supervisory Control Reference

epos.maxonmotor.com

Document ID: rel6554

maxon motor ag Brünigstrasse 220 P.O.Box 263 CH-6072 Sachseln Phone +41 41 666 15 00 Fax +41 41 666 16 50 www.maxonmotor.com

PLEASE READ THIS FIRST

These instructions are intended for qualified technical personnel. Prior commencing with any activities …

- *you must carefully read and understand this manual and*
- *you must follow the instructions given therein.*

We have tried to provide you with all information necessary to install and commission the equipment in a **secure**, **safe** and **time-saving** manner. Our main focus is …

- to familiarize you with all relevant technical aspects,
- to let you know the easiest way of doing,
- to alert you of any possibly dangerous situation you might encounter or that you might cause if you do not follow the description,
- to **write as little** and to **say as much** as possible and
- not to bore you with things you already know.

Likewise, we tried to skip repetitive information! Thus, you will find things **mentioned just once**. If, for example, an earlier mentioned action fits other occasions you then will be directed to that text passage with a respective reference.

Follow any stated reference – observe respective information – then go back and continue with the task!

PREREQUISITES FOR PERMISSION TO COMMENCE INSTALLATION

The EPOS2 P is considered as partly completed machinery according to EU directive 2006/42/EC, Article 2, Clause (g) and therefore **is intended to be incorporated into or assembled with other machinery or other partly completed machinery or equipment**.

You must not put the device into service, …

- *unless you have made completely sure that the other machinery – the surrounding system the device is intended to be incorporated to – fully complies with the requirements stated in EU directive 2006/42/EC!*
- *unless the surrounding system fulfills all relevant health and safety aspects!*
- *unless all respective interfaces have been established and fulfill the stated requirements!*

TABLE OF CONTENTS

1 About this Document

1.1 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. maxon motor, 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.

1.2 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.

1.3 How to use

Take note of the following notations and codes which will be used throughout the document.

Notation	Explanation
«Abcd»	indicating a title or a name (such as of document, product, mode, etc.)
(n)	referring to an item (such as order number, list item, etc.)
➔	denotes “see”, “see also”, “take note of” or “go to”

Table 1-1 Notations used in this Document

For CAN-specific notations, abbreviations and terms ➔page 10-52

1.4 Symbols and Signs

1.4.1 Informatory Signs

Requirement / Note / Remark
Indicates an action you must perform prior continuing or refers to information on a particular item.

Best Practice
Gives advice on the easiest and best way to proceed.

Material Damage
Points out information particular to potential damage of equipment.

1.5 Sources for additional Information

For further details and additional information, please refer to below listed sources:

#	Reference
[1]	CiA 301 Communication Profile for Industrial Systems www.can-cia.org
[2]	CiA 402 Device Profile for Drives and Motion Control www.can-cia.org
[3]	CiA 305 Layer Setting Services (LSS) and Protocols www.can-cia.org
[4]	CiA 306 Electronic Data Sheet Specification www.can-cia.org
[5]	Bosch's CAN Specification 2.0 www.can-cia.org
[6]	USB Implementers Forum: Universal Serial Bus Revision 2.0 Specification www.usb.org/developers/docs
[7]	Konrad Etschberger: Controller Area Network ISBN 3-446-21776-2

Table 1-2 Sources for additional Information

1.6 Trademarks and Brand Names

For easier legibility, registered brand names are listed below and will not be further tagged with their respective trademark. It must be understood that the brands (the below list is not necessarily concluding) are protected by copyright and/or other intellectual property rights even if their legal trademarks are omitted in the later course of this document.

Brand Name	Trademark Owner
CANopen® CiA®	© CiA CAN in Automation e.V, DE-Nuremberg
Windows®	© Microsoft Corporation, USA-Redmond, WA

Table 1-3 Brand Names and Trademark Owners

1.7 Copyright

maxon motor ag
Brünigstrasse 220
P.O.Box 263
CH-6072 Sachseln
Switzerland

Phone +41 41 666 15 00
Fax +41 41 666 16 50

www.maxonmotor.com

••page intentionally left blank••

2 Introduction

The present document provides you with the supervisory control communication interfaces details on the EPOS2 P Programmable Positioning Controllers. It contains…

- the supervisory control system description,
- explanations on how to get started (as an example),
- the supervisory variable access,
- the program download control, and
- the communication description.

Find the latest edition of the present document, as well as additional documentation and software to the EPOS2 P Programmable Positioning Controllers also on the Internet: ➔www.maxonmotor.com

2.1 Documentation Structure

The present document is part of a documentation set. Please find below an overview on the documentation hierarchy and the interrelationship of its individual parts:

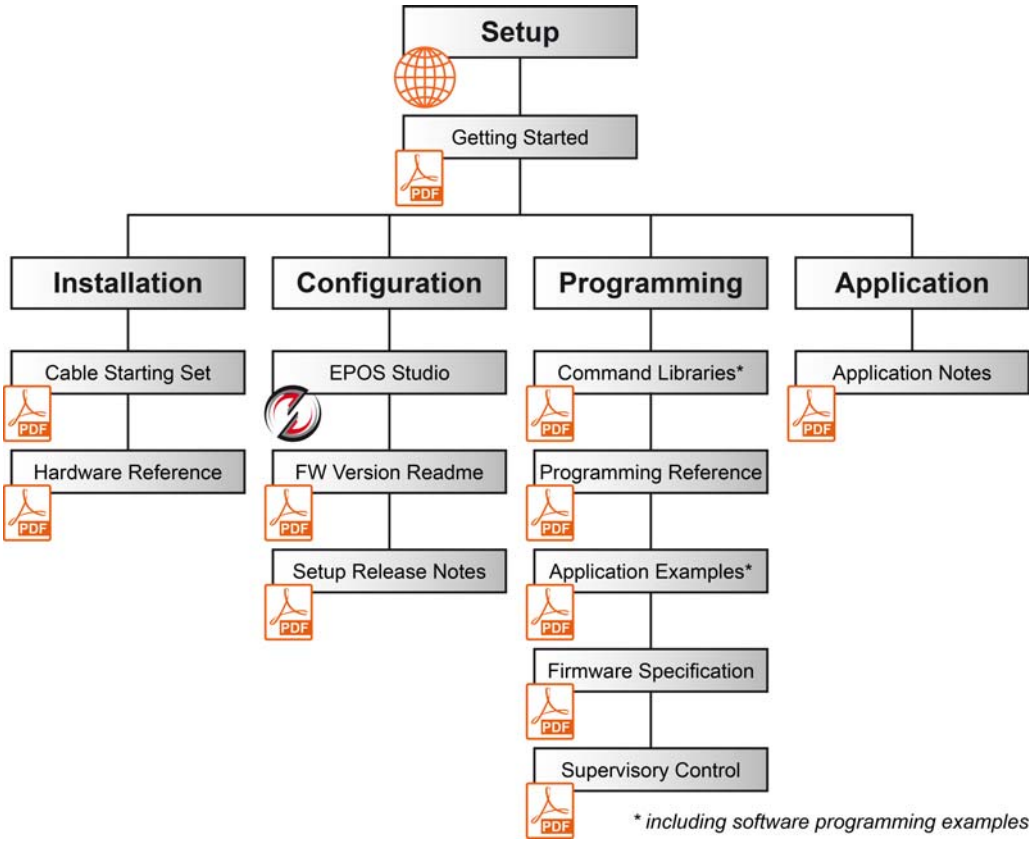


Figure 2-1 Documentation Structure

••page intentionally left blank••

3 System Description

3.1 System Overview

The following charts show the typical topology of a supervisory control system:

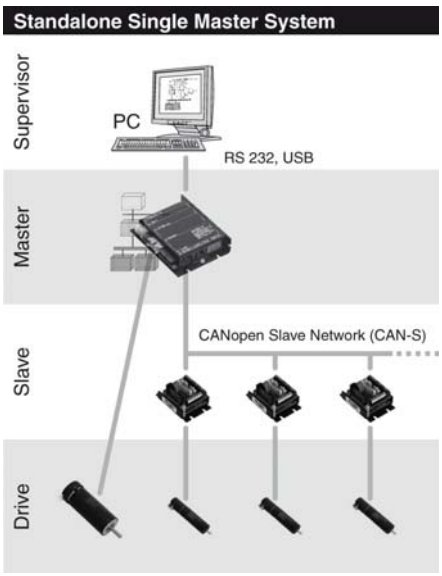


Figure 3-2 Supervisory Control System – Single Master System

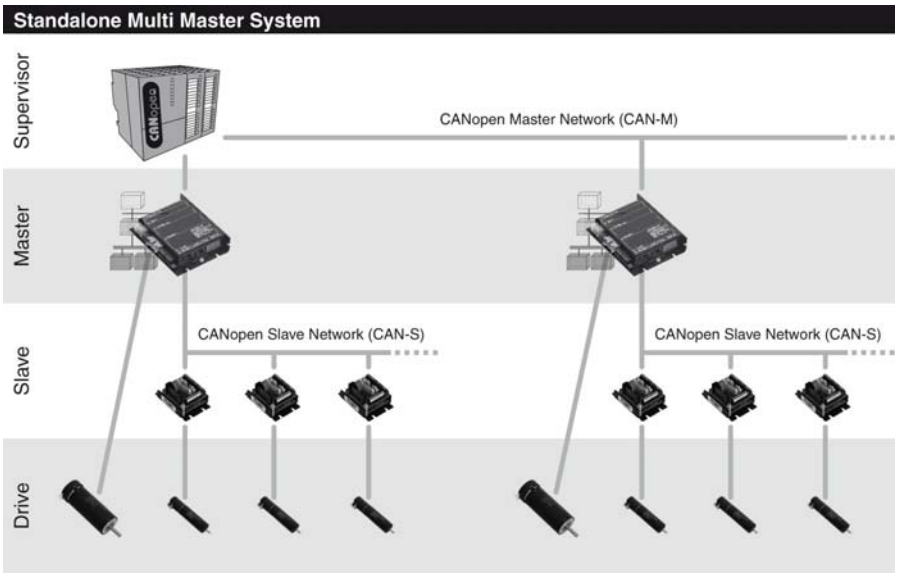


Figure 3-3 Supervisory Control System – Multi Master System

3.2 Network Topology

Supervisor Level (process level)

The supervisor can perform the following tasks: Commanding and observing the master, as well as dispense management and data acquisition. Typically, a PC or a powerful PLC will be used.

Master Level (PLC level)

The master is responsible to control and regulate the working sequence. The EPOS2 P is used in this level.

Slave Level (motion control / field level)

The EPOS2 Positioning Controller may be used to control the motor.

Drive Level (actuator level)

Brushed DC and brushless DC motors, as well as inputs and outputs will be used.

3.3 Communication

The supervisor control system (master device) and the commissioning tools («EPOS Studio» OpenPCS or other programming tools) communicate over RS232, USB and/or CAN-M interface with the EPOS2 P.

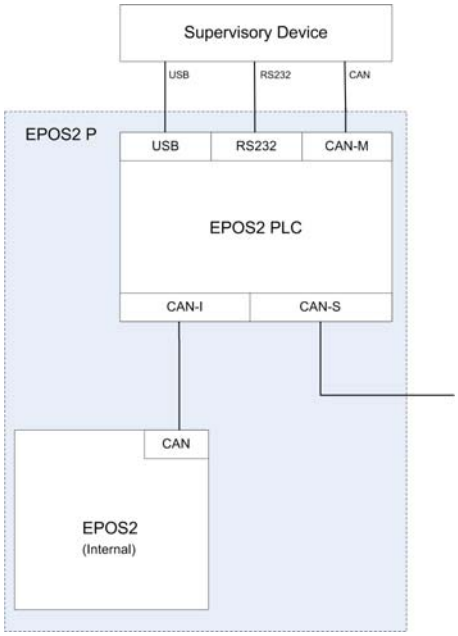


Figure 3-4 Communication

3.3.1 Interfaces

Tool	Interface		
	RS232	USB	CAN
EPOS Studio	X	X	X
OpenPCS	X	X	X

Table 3-4 Interfaces & Tools

Supervisor	Interface			
	RS232	USB	CAN	Driver available
Windows PC	X	X	X	X*1)
PLC	X	X	X	–
Microcontroller	X	X	X	–
Remark: *1) For detailed information on drivers ➔separate document «EPOS2 P Windows 32-Bit DLL».				

Table 3-5 Interfaces & Supervisor/Driver

3.3.2 Gateways

To configure slaves (EPOS2) from the supervisor level, gateway functionalities are available on the master controller (EPOS2 P):

- USB → CAN-S / CAN-I gateway
- RS232 → CAN-S / CAN-I gateway
- CAN-M → CAN-S / CAN-I gateway

For details ➔chapter “10 A3 Communication Guide” on page 10-41.

3.4 Object Dictionaries and Addressing Scheme

The master controller EPOS2 P features three different object dictionary.

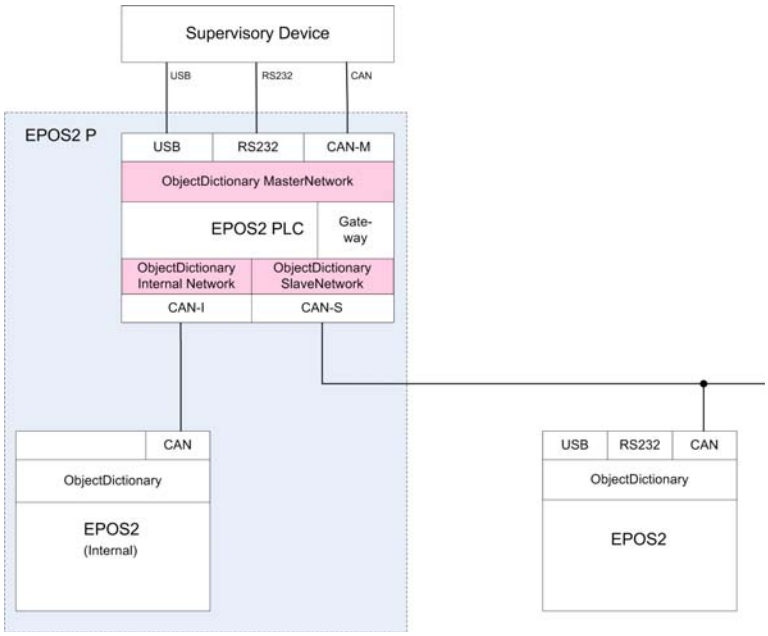


Figure 3-5 Object Dictionaries

Object Dictionary	Description
Master Network	contains objects for CAN-M, USB, RS232: communication to the supervisor
Internal Network	contains Objects for CAN-I internal: used only for communication to the internal EPOS2
Slave Network	contains Objects for CAN-S: Used for external slaves

Table 3-6 Object Dictionaries

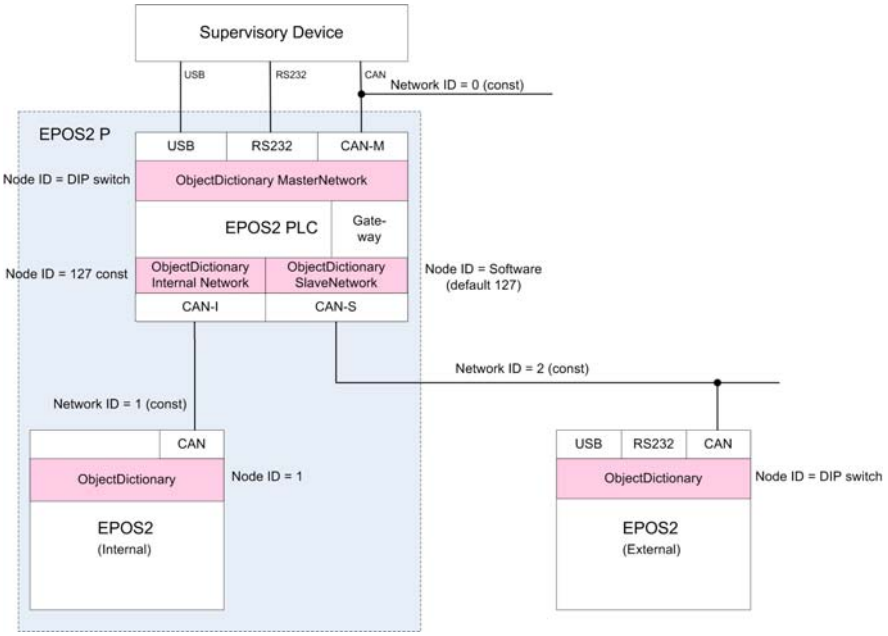


Figure 3-6 Addressing Scheme

Object Dictionary	Network ID (const)	Node ID	Note to Node ID
Master Network	0	DIP switch	configurable
Internal Network	1	127	constant
Slave Network	2	Software	configurable
EPOS 2 (internal)	1	1	constant
EPOS 2 (external)	2	DIP switch	configurable

Table 3-7 Addressing Scheme

3.5 Tools

maxon motor provides tools for commissioning and programming the EPOS2 P.

3.5.1 Commissioning Tool

The «EPOS Studio» is designed for commissioning the EPOS2 P. For detailed information ➔chapter “4 Getting Started” on page 4-17.

3.5.2 Programming Tool

The «OpenPCS» (an integrated part of «EPOS Studio») is used for programming the EPOS2 P. For detailed information on programming ➔separate document «EPOS2 P Programming Reference».

••page intentionally left blank••

4 Getting Started

This chapter explains how to set up communication to a supervisor system by means of an example.

4.1 Drive Setup

First, the EPOS2 P Programmable Positioning Controller must be configured for the drive system (for details ➔respective chapters of the controller's «Getting Started» document).

1) Execute the Startup Wizard in «EPOS Studio».
2) Perform "Regulation Gains Tuning" in «EPOS Studio».
3) Repeat above steps for each individual CAN slave.

4.2 Network Configuration

For communicating between the supervisor system and the EPOS2 P, the communication channels must be enabled.
The process variables used for the communication between the supervisor and the master must be set using the network configuration tool (for details ➔respective chapters of separate document «EPOS2 P Programming Reference»):

Best Practice
It will be helpful to use the same variable names as in the IEC 61131 program.

1) Add the process variables needed for the communication to the supervisor.

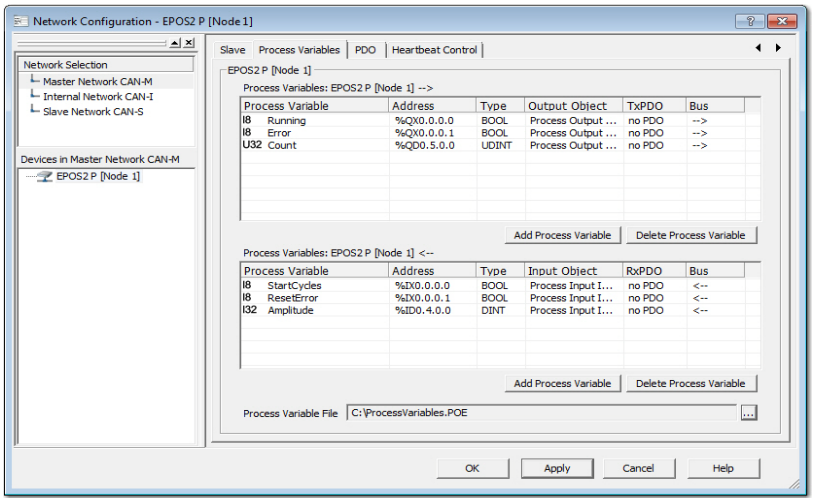


Figure 4-7 Network Configuration

2) A process variable file will be created. You may view, save or print the file.

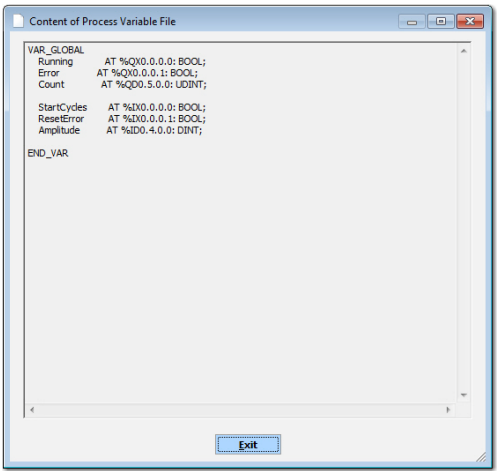


Figure 4-8 Content of Process Variable File

3) If you save the file as “ProcessVariables.poe” in the IEC 61131 program folder, you may directly use it later on as process variable file in the IEC program.

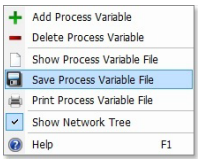


Figure 4-9 Save Process Variable File

4.3 IEC 61131 Program

This chapter shows how to implement the communication between the supervisor system and the EPOS2 P in the IEC 61131 program (for basic information on IEC programming with the EPOS2 P ➔respective chapters of separate document «EPOS2 P Programming Reference»).

The earlier generated process variable file (➔chapter “4.2 Network Configuration” on page 4-17) can be saved as “ProcessVariables.POE” in the IEC 61131 program folder for direct use in the program.

1) Open “ProcessVariables.POE” from the IEC 61131 program folder.

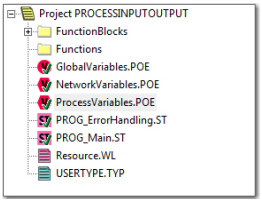


Figure 4-10 IEC 61131 Program

2) Open “ProcessVariables.POE” using the «OpenPCS» programming tool. You may edit or add comments.

```
(*****************************************************************************************************************
**  maxon motor ag
**  All rights reserved
******************************************************************************************************************
**  Project     :  ProcessInputOutput
**  File        :  GlobalVariables
**  Description :  Definition of Global Variables
**  Creation    :  17.02.2010, ODR
**  Modification:  17.02.2010, ODR, Initial Version
******************************************************************************************************************)

VAR_GLOBAL
    Axis0               : AXIS_REF     := (AxisNo := 0);       (* Axis definition *)
    eStateMain          : MAIN_STATE;                          (* Process-State *)
    eStateErrorHandling : EH_STATE;                            (* ErrorHandling-State *)

    Main_ErrorFlag      : BOOL;                               (* Error in a Process-State *)
    Main_ErrorID        : DINT;                               (* ErrorID from any Function Block *)
    Main_ErrorState     : MAIN_STATE;                         (* Error-State *)

    Axis0_ErrorFlag     : BOOL;                               (* Axis0 Error *)
    Axis0_ErrorCode     : DINT;                               (* Error Code from Axis0 *)
END_VAR
```

Figure 4-11 ProcessVariables.POE

3) Declare the process variables in the variable declaration section (red frame) of the respective program as shown:

```
(*****************************************************************************************************************
**  maxon motor ag
**  All rights reserved
******************************************************************************************************************
**  Project     :  ProcessInputOutput
**  File        :  Main State Machine
**  Description :  Implementation of Main State Machine
**  Creation    :  17.02.2010, ODR
**  Modification:  17.02.2010, ODR, Initial Version
******************************************************************************************************************)

VAR_EXTERNAL
    (* Axis Defintion *)
    Axis0                           : AXIS_REF;

    (* State Machines *)
    eStateMain                      : MAIN_STATE;
    eStateErrorHandling             : EH_STATE;

    (* Error Main State Machine *)
    Main_ErrorFlag                  : BOOL;                    (* Error in a Process State *)
    Main_ErrorID                    : DINT;                    (* ErrorID from any Function Block *)
    Main_ErrorState                 : MAIN_STATE;              (* Error-State *)

    (* Error Axis0 *)
    Axis0_ErrorFlag                 : BOOL;                    (* Axis0 Error *)
    Axis0_ErrorCode                 : DINT;                    (* Axis0 Error Code *)

    (* Input Process Variables *)
    StartCycles                     : BOOL;                    (* Start/Stop Cycles *)
    Amplitude                       : DINT;                    (* Cycle Amplitude *)

    (* Output Process Variables *)
    Running                         : BOOL;                    (* Cycle Running *)
    Count                           : UDINT;                   (* Cycle Count *)
END_VAR
```

Figure 4-12 Variable Declaration

4) Now, the declared variables can be used in the IEC program like any other variable.

```
IF eStateMain = MAIN_Stopped THEN

    (* Power Off *)
    fbPower(Axis := Axis0, Enable := FALSE);

    (* Start Cycles Trigger *)
    StartCyclesTrigger.CLK := StartCycles;
    StartCyclesTrigger();
    IF StartCyclesTrigger.Q THEN
        Count := 0;
        Running := TRUE;
        eStateMain := MAIN_InitCycle;
        StartCyclesTrigger();
    END_IF;

    (* ErrorHandling *)
    IF eStateErrorHandling = EH_ErrorReaction THEN
        eStateMain := MAIN_Error;
    END_IF;
```

Figure 4-13 Use of declared Variables

4.4 Supervisory Program

The following example of a supervisory program shows how to command the EPOS2 P from a PC as supervisor system.

1) Define communication setting:
   a) Select desired communication channel (CANopen or MAXON SERIAL V2).
   b) Define Interface Name, Port Name, Baudrate and Timout.

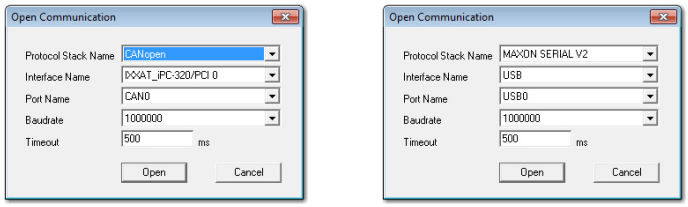


Figure 4-14 Supervisory Program Communication Channel

2) Set HMI functions (for details ➔Table 4-8) for the communication with the EPOS2 P:

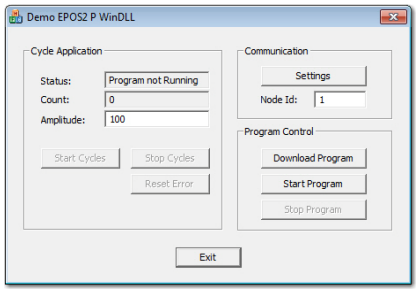


Figure 4-15 EPOS2 P WinDLL

Area	Item	Description
Communication	Settings	to change the communication settings (the Open Communication window of the previous step will appear)
	Node Id	to define the EPOS2 P Node Id
Program Control	Download Program	to select an IEC 61131 program and to download it to the EPOS2 P
	Start Program	to start the downloaded IEC 61131 program on the EPOS2 P
	Stop Program	to stop the running IEC 61131 program in the EPOS2 P
Cycle Application	Status	displays the current program status
	Count	displays the current number of cycles
	Amplitude	to set the motion amplitude in quad counts [qc]
	Start Cycles	to start the execution of the cycles
	Stop Cycles	to stop the execution of the cycles
	Reset Error	to reset errors
Button	Exit	to close the supervisory program

Table 4-8 Supervisory Program – Functions

5 Network Configuration

Remark
This section describes only the configuration of the "Master Network CAN-M". For further information ➔separate document «EPOS2 P Programming Reference».

The network configuration tool is used to configure the communication of a supervisory controller to the "Master Network Object Dictionary".

5.1 Slave

The tab "Slave" displays the current communication settings of the EPOS2 P as a member of the "Master Network CAN-M".

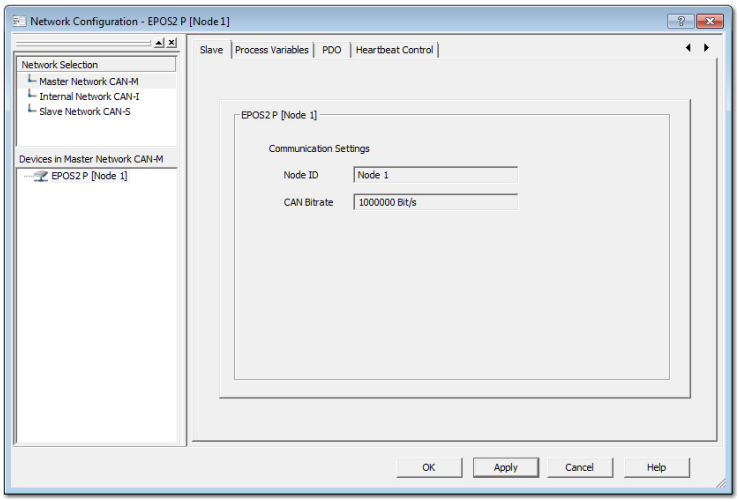


Figure 5-16 Slave Tab

Area	Item	Description
Context Menu	Show Network Tree	to show/hide the Network Selection Window
	Set Default	to restore the default configuration of the selected device
	Help	to display help information on the network configuration

Table 5-9 Slave Tab – Functions

5.2 Process Variables

The tab “Process Variables” is used to configure mapping of process output/input objects to process variables using the IEC 61131 program.

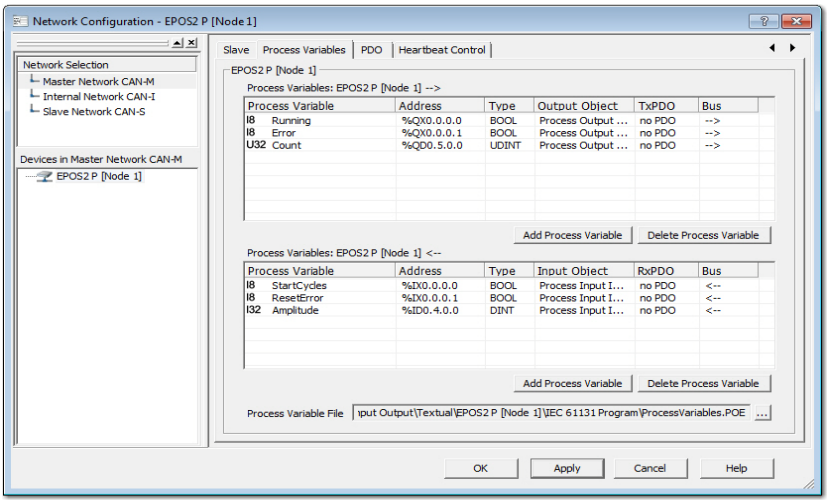


Figure 5-17 Process Variables Tab

Area	Item	Description
Table Columns	Process Variable	name of process variable to be used in IEC 61131 program
	Address	direct address of the process variable in IEC 61131 program (for example Running AT %QX 0.0.0.0: BOOL)
	Type	IEC 61131 type of process variable
	Output Object Input Object	object in “Master Network” object dictionary, may be mapped to a PDO
	TxPDO RxPDO	configured transmit/receive PDO to exchange data with the supervisory controller – PDO mapping of an output/ input object is optional (➔Table 5-11, “PDO Mapping”)
	Bus	direction of the data exchange
Buttons	Add Process Variable	to add a new process variable to the table
	Delete Process Variable	to delete an existing process variable from the table
Text Field	Process Variable File	to specify a process variable file (*.poe), which can be imported to the programming tool «OpenPCS»

Table 5-10 Process Variables Tab – Functions

5.2.1 Add Process Variable

The dialog “Add Process Variable” displays the configuration options for a new process variable.

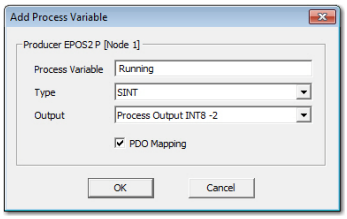


Figure 5-18 Add Process Variable Dialog

Area	Item	Description
Output	Process Variable	name of process variable used in IEC 61131 program – a default value will be suggested during start-up of the dialog and may be changed
	Type	IEC 61131 type of process variable – if you select type “BOOL”, you must define a bit number
	Output Object Input Object	object in “Master Network” object dictionary
	PDO Mapping	**Checked:** Object will automatically be mapped to a PDO **Not Checked:** Object will not be mapped to a PDO

Table 5-11 Add Process Variable Dialog – Functions

5.2.2 Context Menu

Area	Item	Description
Context Menu	Add Process Variable	to add a new process variable to the table
	Delete Process Variable	to delete an existing process variable from the table
	Show Process Variable File	to display the process variable declarations
	Save Process Variable File	to save the process variable declarations
	Print Process Variable File	to print the process variable declarations
	Help	to display help information on the network configuration

Table 5-12 Process Variables – Context Menu Functions

5.3 PDO

The tab “PDO” is used to edit and change the PDO configuration of the Master Network.

Configuration of process variables automatically adds PDOs and PDO Mappings
Make sure not to destroy the PDO configuration of a process variable!

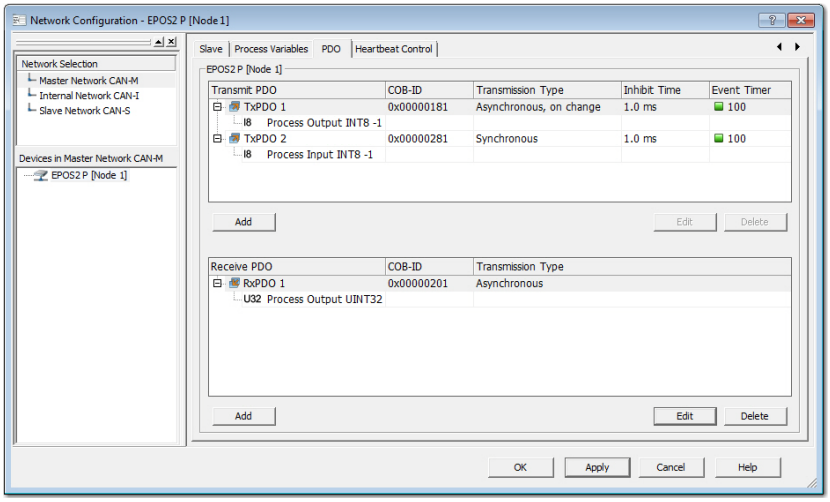


Figure 5-19 PDO Tab

Area	Item	Description
Table Columns	Transmit PDO Receive PDO	PDOs and mapped object of the PDO
	COB-ID	11-Bit Identifier used by the PDO
	Transmission Type	defines the transmission/reception character of a PDO
	Inhibit Time	minimal transmission interval for asynchronous PDOs **Note!** An inhibit time of “0” (zero) represents a potential risk for bus overload!
	Event Timer	elapsed timer to trigger the asynchronous PDO transmission
Buttons	Add	to add a new Transmit/Receive PDO to the list **Note!** if inactive, no more PDOs can be added
	Edit	to change settings of an existing PDO
	Delete	to delete an existing PDO from the list

Table 5-13 PDO Tab – Functions

5.3.1 Edit

The dialog “Edit” displays the configuration options for Transmit and Receive PDOs.

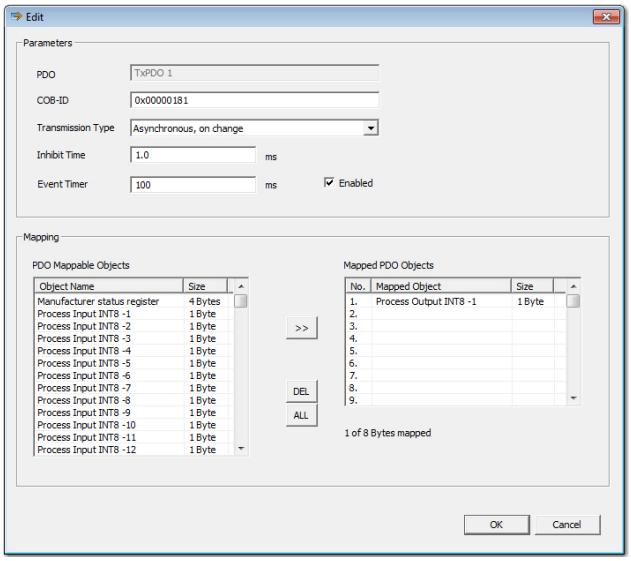


Figure 5-20 Edit Dialog

Area	Item	Description
Parameters	PDO	name of PDO being configured
	COB-ID	11-Bit Identifier used by the PDO
	Transmission Type	defines the transmission/reception character of a PDO **Asynchronous:** PDO transmission is triggered by value change or event timer **Asynchronous RTR only:** PDO can be requested by a remote transfer request **Synchronous:** PDO transmission is triggered by the Sync Master
	Inhibit Time	minimal transmission interval for asynchronous PDOs **Note!** An inhibit time of “0” (zero) represents a potential risk for bus overload!
	Event Timer	elapsed timer to trigger the asynchronous PDO transmission
Mapping	PDO Mappable Objects	list of all objects that can be mapped to a PDO
	Mapped PDO Objects	list of all objects that are mapped to the PDO
Buttons	>>	to add an object to the PDO mapping
	DEL	to delete an object from the PDO mapping
	ALL	to delete all objects from the PDO mapping

Table 5-14 Edit Dialog – Functions

5.4 Heartbeat Control

The tab “Heartbeat Control” is used to configure the error control service.

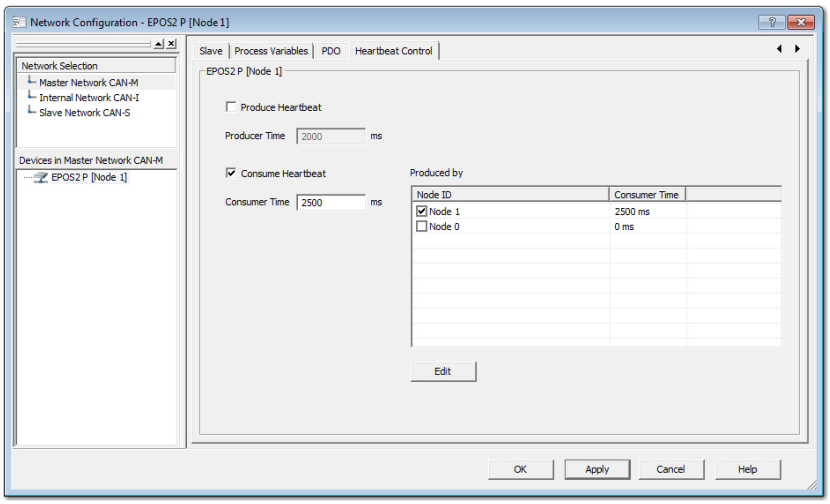


Figure 5-21 Heartbeat Control Tab

Area	Item	Description
Node	Produce Heartbeat	to activate the heartbeat producer
	Producer Time	cycle time of the heartbeat message
	Consume Heartbeat	to activate the heartbeat consumer
	Consumer Time	monitoring time of the heartbeat message
	Produced by	to activate/edit the node to be monitored

Table 5-15 Heartbeat Control Tab – Functions

6 IEC 61131 Program

The program is used to address network devices using network variables as described in ➔chapter “5 Network Configuration” on page 5-21.

Perform the following steps to import the “Process Variable File” into the IEC 61131 program.

1) Open the IEC 61131 program in the «OpenPCS» programming tool.
2) Select the menu item “Import” in menu “File”, submenu “File”.
3) Open context menu (right click) and click menu item “Link to Active Resource” to use the network variables.

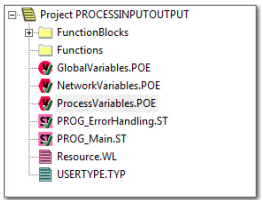


Figure 6-22 IEC 61131 – Context Menu

4) The file “ProcessVariables.poe” will appear similar as to following sample. The variables may be used in the main program.

```
(*****************************************************************************************************
**  maxon motor ag
**  All rights reserved
*****************************************************************************************************
**  Project     :  ProcessInputOutput
**  File        :  GlobalVariables
**  Description :  Definition of Global Variables
**  Creation    :  17.02.2010, ODR
**  Modification:  17.02.2010, ODR, Initial Version
*****************************************************************************************************)

VAR_GLOBAL
    Axis0               : AXIS_REF     := (AxisNo := 0);       (* Axis definition *)
    eStateMain          : MAIN_STATE;                          (* Process-State *)
    eStateErrorHandling : EH_STATE;                            (* ErrorHandling-State *)

    Main_ErrorFlag      : BOOL;                               (* Error in a Process-State *)
    Main_ErrorID        : DINT;                               (* ErrorID from any Function Block *)
    Main_ErrorState     : MAIN_STATE;                         (* Error-State *)

    Axis0_ErrorFlag     : BOOL;                               (* Axis0 Error *)
    Axis0_ErrorCode     : DINT;                               (* Error Code from Axis0 *)
END_VAR
```

Figure 6-23 ProcessVariables.poe (Sample)

5) Import the direct variables into the external declaration section of the IEC 61131 program.

```
(*****************************************************************************************************
**  maxon motor ag
**  All rights reserved
*****************************************************************************************************
**  Project     :  ProcessInputOutput
**  File        :  Main State Machine
**  Description :  Implementation of Main State Machine
**  Creation    :  17.02.2010, ODR
**  Modification:  17.02.2010, ODR, Initial Version
*****************************************************************************************************)

VAR_EXTERNAL
    (* Axis Defintion *)
    Axis0                           : AXIS_REF;

    (* State Machines *)
    eStateMain                      : MAIN_STATE;
    eStateErrorHandling             : EH_STATE;

    (* Error Main State Machine *)
    Main_ErrorFlag                  : BOOL;                    (* Error in a Process State *)
    Main_ErrorID                    : DINT;                    (* ErrorID from any Function Block *)
    Main_ErrorState                 : MAIN_STATE;              (* Error-State *)

    (* Error Axis0 *)
    Axis0_ErrorFlag                 : BOOL;                    (* Axis0 Error *)
    Axis0_ErrorCode                 : DINT;                    (* Axis0 Error Code *)

    (* Input Process Variables *)
    StartCycles                     : BOOL;                    (* Start/Stop Cycles *)
    Amplitude                       : DINT;                    (* Cycle Amplitude *)

    (* Output Process Variables *)
    Running                         : BOOL;                    (* Cycle Running *)
    Count                           : UDINT;                   (* Cycle Count *)
END_VAR
```

Figure 6-24 Direct Variables

6) You now may use the variable in the IEC 61131 program as to following sample.

```
IF eStateMain = MAIN_Stopped THEN

    (* Power Off *)
    fbPower(Axis := Axis0, Enable := FALSE);

    (* Start Cycles Trigger *)
    StartCyclesTrigger.CLK := StartCycles;
    StartCyclesTrigger();
    IF StartCyclesTrigger.Q THEN
        Count := 0;
        Running := TRUE;
        eStateMain := MAIN_InitCycle;
        StartCyclesTrigger();
    END_IF;

    (* ErrorHandling *)
    IF eStateErrorHandling = EH_ErrorReaction THEN
        eStateMain := MAIN_Error;
    END_IF;
```

Figure 6-25 IEC 61131 – imported Variables (Sample)

7 Supervisory Variable Access

7.1 Single Process Variable

The supervisory controller can read and write process variables via the USB, RS232 or CAN network.

The “Master Network Object Dictionary” provides the possibility to access process variables by addressing the process objects.

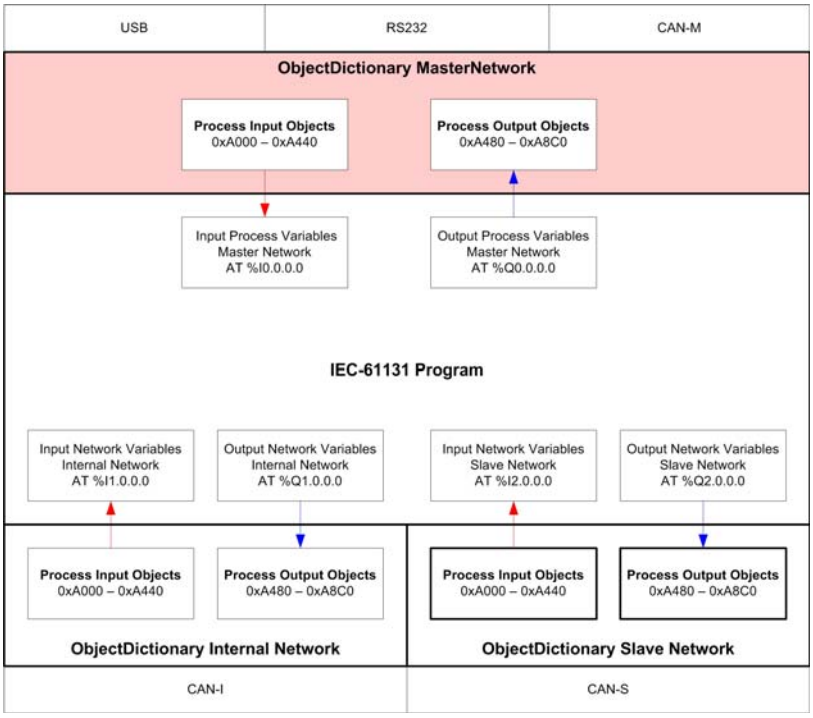


Figure 7-26 Single Process Variable

7.1.1 Master Network “Process Input Objects”

The following list defines the available process input objects and the corresponding input process variables:

#	Type	Process Input Object		Process Input Variable	
		Index	Subindex	Address	Description
16	SINT	0xA000	0x01…0x10	AT %IB 0.0.0.0 - 0.0.15.0	Signed 8-Bit object
16	USINT	0xA040	0x01…0x10	AT %IB 0.1.0.0 - 0.1.15.0	Unsigned 8-Bit object
16	INT	0xA0C0	0x01…0x10	AT %IW 0.2.0.0 - 0.2.30.0	Signed 16-Bit object
16	UINT	0xA100	0x01…0x10	AT %IW 0.3.0.0 - 0.3.30.0	Unsigned 16-Bit object
16	DINT	0xA1C0	0x01…0x10	AT %ID 0.4.0.0 - 0.4.60.0	Signed 32-Bit object
16	UDINT	0xA200	0x01…0x10	AT %ID 0.5.0.0 - 0.5.60.0	Unsigned 32-Bit object

Table 7-16 Master Network – Process Input Objects

7.1.2 Master Network “Process Output Objects”

The following list defines the available process output objects and the corresponding output process variables.

#	Type	Process Output Object		Process Output Variable	
		Index	Subindex	Address	Description
16	SINT	0xA480	0x01…0x10	AT %QB 0.0.0.0 - 0.0.15.0	Signed 8-Bit object
16	USINT	0xA4C0	0x01…0x10	AT %QB 0.1.0.0 - 0.1.15.0	Unsigned 8-Bit object
16	INT	0xA540	0x01…0x10	AT %QW 0.2.0.0 - 0.2.30.0	Signed 16-Bit object
16	UINT	0xA580	0x01…0x10	AT %QW 0.3.0.0 - 0.3.30.0	Unsigned 16-Bit object
16	DINT	0xA640	0x01…0x10	AT %QD 0.4.0.0 - 0.4.60.0	Signed 32-Bit object
16	UDINT	0xA680	0x01…0x10	AT %QD 0.5.0.0 - 0.5.60.0	Unsigned 32-Bit object

Table 7-17 Master Network – Process Input Objects

7.2 Multi Process Image Variables

7.2.1 Master Network “Process Image Object”

The supervisory controller can read and write several process variables in one block.

Using the “Process Image Object”, fast updating between the supervisory controller and the IEC 61131 program via the USB, RS232 or CAN network is possible.

The object “Process Image Range Inputs” defines the data range of the input process image.

Index	Subindex	Description	Data Byte	Access
0x2F70	0x01	Process Image Range Inputs	UDINT	RW

Table 7-18 Process Image Range Inputs – Description

Bit	Range	Description
0…15	0x0000…0x00DF	Selection Start
16…31	0x0000…0x00E0	Selection Length

Table 7-19 Process Image Range Inputs – Bits

Write several input process variables in one block.

#	Type	Selection		Process Input Variable	
		Start	Length	Address	Description
16	SINT	0x0000	0x0010	AT %IB 0.0.0.0 - 0.0.15.0	Signed 8-Bit
16	USINT	0x0010	0x0010	AT %IB 0.1.0.0 - 0.1.15.0	Unsigned 8-Bit
16	INT	0x0020	0x0020	AT %IW 0.2.0.0 - 0.2.30.0	Signed 16-Bit
16	UINT	0x0040	0x0020	AT %IW 0.3.0.0 - 0.3.30.0	Unsigned 16-Bit
16	DINT	0x0060	0x0040	AT %ID 0.4.0.0 - 0.4.60.0	Signed 32-Bit
16	UDINT	0x00A0	0x0040	AT %ID 0.5.0.0 - 0.5.60.0	Unsigned 32-Bit

Table 7-20 Master Network – Process Image Object (Inputs)

Selection length set to 0x0000 represents access on all process input variables, regardless of the selection start value.

The object “Process Image Range Outputs” defines the data range of the output process image.

Index	Subindex	Description	Data Byte	Access
0x2F70	0x02	Selected Range Outputs	UDINT	RW

Table 7-21 Process Image Range Outputs – Description

Bit	Range	Description
0…15	0x0000…0x00DF	Selection Start
16…31	0x0000…0x00E0	Selection Length

Table 7-22 Process Image Range Outputs – Bits

Read several output process variables in one block.

#	Type	Selection		Process Output Variable	
		Start	Length	Address	Description
16	SINT	0x0000	0x0010	AT %QB 0.0.0.0 - 0.0.15.0	Signed 8-Bit
16	USINT	0x0010	0x0010	AT %QB 0.1.0.0 - 0.1.15.0	Unsigned 8-Bit
16	INT	0x0020	0x0020	AT %QW 0.2.0.0 - 0.2.30.0	Signed 16-Bit
16	UINT	0x0040	0x0020	AT %QW 0.3.0.0 - 0.3.30.0	Unsigned 16-Bit
16	DINT	0x0060	0x0040	AT %QD 0.4.0.0 - 0.4.60.0	Signed 32-Bit
16	UDINT	0x00A0	0x0040	AT %QD 0.5.0.0 - 0.5.60.0	Unsigned 32-Bit

Table 7-23 Master Network – Process Image Object (Outputs)

Selection length set to 0x0000 represents access on all process output variables, regardless of the selection start value.

The object “Process Image Domain” provides access to the process variables as configured with “Process Image Range Inputs”, respectively “Process Image Range Outputs”.

Index	Subindex	Description	Data Byte	Access
0x2F70	0x03	Process Image Domain	DOMAIN	RW

Table 7-24 Process Image Domain – Description

••page intentionally left blank••

8 A1 Program Download & Control

The supervisory controller can download and control an IEC 61131 program via the USB, RS232 or CAN network.

The “Master Network Object Dictionary” provides the possibility to download and control a IEC 61131 program by addressing objects.

8.1 How to handle a Program Download via Supervisory Controller

Execute the following action to download an IEC 61131 program via supervisory controller.

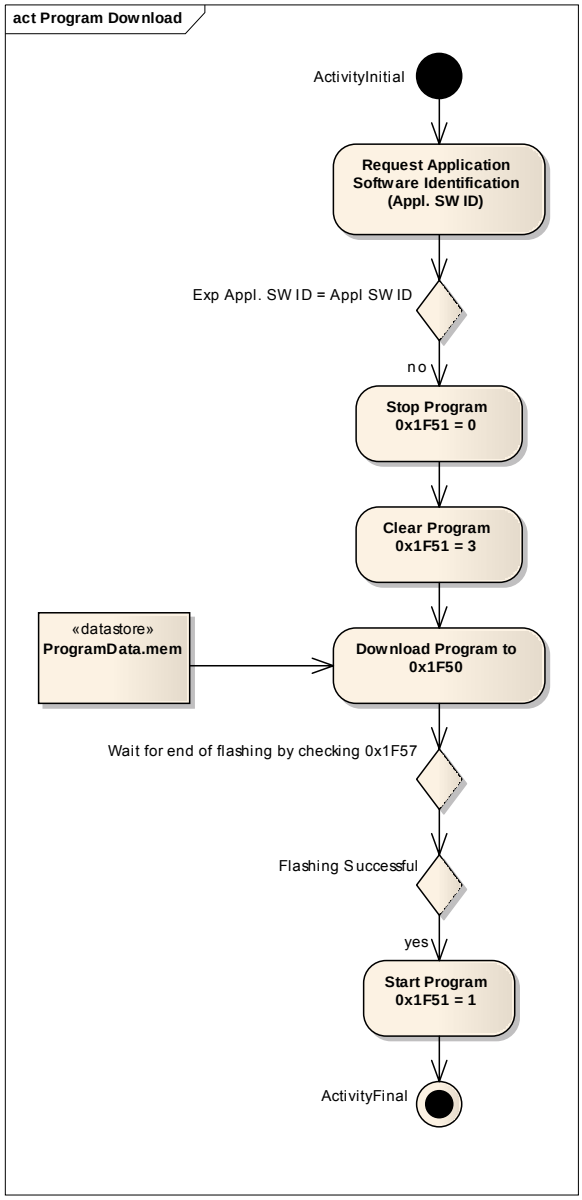


Figure 8-27 Downloading an IEC 61131 Program

8.2 Object «Program Download»

Used for program download.

Name	Program Data
Index	0x1F50
Number of entries	0x01

Description

The object sends an IEC 61131 program to the EPOS2 P runtime system.

Remarks

- Access is not permitted as long as communication with OpenPCS programming system is established.
- Download is only permitted after the program was cleared.
- The application software identification in the header verifies the program.

Name	Program Number 1
Index	0x1F50
Subindex	0x01
Type	DOMAIN
Access	WO

8.2.1 Program Data Format

The IEC 61131 program file “ProgramData.mem” includes the program code. The program data object 0x1F50-01 is used for the program code download and is located in the subdirectory of the OpenPCS project …\Gen\Resource\ProgramData.mem.

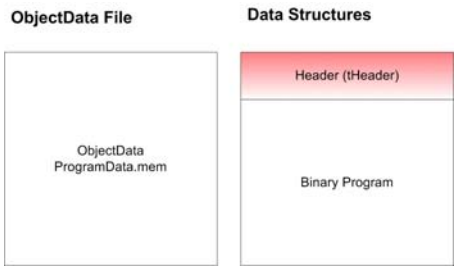


Figure 8-28 Object “Program Download” – Program Data Format

The application software identification in the header verifies the program.

tHeader Data Structure

```
typedef struct
{
UDINT    applicationSoftwareID;          //Date and Time of Program Build
UINT     hardwareVersion;                /Hardware Version
UINT     applicationNumber;              //Application Number
UINT     softwareVersion;                //Software Version
UINT     applicationVersion;             //Application Version
UINT     applicationSoftwareCRC;         //CRC Value (without header)
} tHeader;
```

Remark
The byte order of the data is Little Endian (least significant byte first).

8.3 Object «Application Software Identification»

Used to verify the version of the program and to check if a re-download will be necessary.

Name	Application Software Identification
Index	0x1F56
Number of entries	1

Description

The object can be used to verify Program Number 1. It contains date and time of the program build.

Remarks

- If no valid program is available, the object returns value <0>.

Name	Program Number 1
Index	0x1F56
Subindex	0x01
Type	UNSIGNED32
Access	RO

8.4 Object «Program Control»

Used to control a IEC-61131 program.

Name	Program Control
Index	0x1F51
Number of entries	1

Description

The object controls an IEC 61131 program on the EPOS2 P runtime system.

Remarks

- Write access is not permitted as long as communication with OpenPCS programming system is established or no valid program is available on the device.
- A start or clear command is only permitted in program status “Stopped”.
- A stop command is only permitted in program status “Started”.

Name	Program Number 1
Index	0x1F51
Subindex	0x01
Type	UNSIGNED8
Access	RW

Write Access

Value	Description
0x00	Stop Program
0x01	Start Program (same command as “Cold Start”)
0x02	Reset Program (Reset Device)
0x03	Clear Program (Erase)
0x81	Cold Start (manufacturer-specific)
0x82	Warm Start (manufacturer-specific)
0x83	Hot Start (manufacturer-specific)

Table 8-25 Program Control – Write Access

Read Access

Value	Description
0x00	Program Stopped
0x01	Program Started
0x02	Program Stopped (same read access as value 0x00)
0x03	No Program available

Table 8-26 Program Control – Read Access

8.5 Object «Flash Status Identification»

Name	Flash Status Identification
Index	0x1F57
Number of entries	1

Description

Used to check the flashing process.

Name	Program Number 1
Index	0x1F57
Subindex	0x01
Type	UNSIGNED32
Access	RO

Bit	Value	Description
0	0	Status OK
	1	Download in progress
1…7	0	No error
	1	No valid program
	2	Data format unknown
	3	Data format error or data CRC error
	4	Flash not cleared before write
	5	Flash write error
	6	General address error
	7	Flash secured (write access not permitted)
	64	General internal error
	65…127	Manufacturer-specific (reserved)
8…15		Reserved (always “0”)
16	0	Initialization of program download inactive
	1	Initialization of program download active
17…31		Manufacturer-specific (reserved)

Table 8-27 Flash Status Identification – Bits

••page intentionally left blank••

9 A2 Master Network Object Dictionary

Following table is an overview of all Master Network Object Dictionary Objects. For descriptions of object not listed ➔separate document «EPOS2 P Firmware Specification».

Value	Number of Sub-index	Object Name	Access
0x1000		Device Type	const
0x1001		Error Register	ro
0x1002		Manufacturer Status Register	ro
0x1003		Error History	ro / rw
0x1005		COB-ID SYNC	ro
0x1008		Manufacturer Device Name	const
0x100C		Guard Time	rw
0x100D		Life Time Factor	rw
0x1010		Store Parameters	rw
0x1011		Restore Default Parameters	rw
0x1012		COB-ID TIME	ro
0x1013		High Resolution Time Stamp	rw
0x1014		COB-ID EMCY	ro
0x1016	2	Consumer Heartbeat Time	rw
0x1017	1	Producer Heartbeat Time	rw
0x1018	4	Identity Object	rw
0x1020		Verify Configuration	rw
0x1200		SDO Server Parameter	rw
0x1400…0x1403	4 * 2	RPDO Communication Parameter	rw
0x1600…0x1603	4 * 8	RPDO Mapping Parameter	rw
0x1800…0x1803	4 * 3	TPDO Communication Parameter	rw
0x1A00…0x1A03	4 * 8	TPDO Mapping Parameter	rw
0x1F2A		Lock Current Configuration	rw
0x1F2B		Local Network ID	rw
0x1F2C		Remote Network Routing List	rw
0x1F2E		Functional Elements and Error State	rw
0x1F50	1	Program Data	rw
0x1F51	1	Program Control	rw
0x1F56	1	Application Software Identification	rw
0x1F57	1	Flash Status Identification	rw
0x2000		Node ID	rw
0x2001		CAN Bitrate	rw
0x2002		RS232 Baudrate	rw
0x2003		Version / EPOS2 P Version Numbers	rw
0x2004		Serial Number / Module Version Numbers	rw
0x2005		RS232 Frame Timeout	rw
0x2006		USB Frame Timeout	rw
0x2007		CAN SDO Frame Timeout	rw
0x2009		DIP Switch State	rw
0x200A		CAN Bitrate Display	rw
0x2F51		Startup Program Control	rw

Value	Number of Sub-index	Object Name	Access
0x2F70	3	Process Image	rw
0x2FFF	7	Firmware Download	rw
0x5006		Communication Cycle Period	rw
0x5007		Synchronous Window Length	rw
0x5280…0x529F	32 * 1	Axis 0 to 31 Slave Number (Port, Node-ID)	rw
0xA000	16	Process Input Integer 8	rw
0xA040	16	Process Input Unsigned 8	rw
0xA0C0	16	Process Input Integer 16	rw
0xA100	16	Process Input Unsigned 16	rw
0xA1C0	16	Process Input Integer 32	rw
0xA200	16	Process Input Unsigned 32	rw
0xA480	16	Process Output Integer 8	rw
0xA4C0	16	Process Output Unsigned 8	rw
0xA540	16	Process Output Integer 16	rw
0xA580	16	Process Output Unsigned 16	rw
0xA640	16	Process Output Integer 32	rw
0xA680	16	Process Output Unsigned 32	rw

Table 9-28 A2 Master Network Object Dictionary – Overview

10 A3 Communication Guide

The present section describes the communication between the Supervisor and the Master (EPOS2 P Positioning Controller) via the USB, RS232 and CAN interfaces.

10.1 Command Reference (USB & RS232)

For communication between Supervisor and Master (EPOS2 P), RS232 protocol and USB protocol are identical.

10.1.1 Read Functions

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

«ReadObject»

Read an object value at the given Index and SubIndex.

Request Frame		
OpCode	BYTE	**0x40**
Len	BYTE	3
Parameters	WORD	Network ID
	BYTE	Node ID
	WORD	Index of Object
	BYTE	Subindex of Object

Response Frame		
OpCode	BYTE	**0x00**
Len	BYTE	4
Parameters	DWORD	ErrorCode (➔page 10-70)
	BYTE [4]	Data Bytes Read

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

«InitiateSegmentedRead»

Start reading an object value at the given Index and SubIndex. Use the command “SegmentRead” to read the data.

Request Frame		
OpCode	BYTE	**0x41**
Len	BYTE	3
Parameters	WORD	Network ID
	BYTE	Node ID
	WORD	Index of Object
	BYTE	Subindex of Object

Response Frame		
OpCode	BYTE	**0x00**
Len	BYTE	4
Parameters	DWORD	ErrorCode (➔page 10-70)
	DWORD	Object Data Length (total number of bytes)

«SegmentRead»

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

Request Frame				
OpCode	BYTE	**0x42**		
Len	BYTE	1		
Parameters	BYTE	Toggle not used	[Bit 0] [Bit 1]	Toggle Bit –
	BYTE	Dummy		

Response Frame				
OpCode	BYTE	**0x00**		
Len	BYTE	3…131 (Number of Words)		
Parameters	DWORD	ErrorCode (➔page 10-70)		
	BYTE	Length (max. 255 bytes)		
	BYTE	Toggle LastSegment	[Bit 0] [Bit 1]	Toggle Bit Last Data Segment
	BYTE [0…255]	Data Bytes Read		

10.1.2 Write Functions

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

«WriteObject»

Write an object value at the given Index and SubIndex.

Request Frame		
OpCode	BYTE	**0x48**
Len	BYTE	5
Parameters	WORD	Network ID
	BYTE	Node ID
	WORD	Index of Object
	BYTE	Subindex of Object
	BYTE [4]	Data Bytes to write

Response Frame		
OpCode	BYTE	**0x00**
Len	BYTE	2
Parameters	DWORD	ErrorCode (➔page 10-70)

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

«InitiateSegmentedWrite»

Start writing an object value at the given Index and SubIndex. Use the command “SegmentWrite” to write the data.

Request Frame		
OpCode	BYTE	**0x49**
Len	BYTE	5
Parameters	WORD	Network ID
	BYTE	Node ID
	WORD	Index of Object
	BYTE	Subindex of Object
	DWORD	Object Data Length (total number of bytes)

Response Frame		
OpCode	BYTE	**0x00**
Len	BYTE	2
Parameters	DWORD	ErrorCode (➔page 10-70)

«SegmentWrite»

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

Request Frame				
OpCode	BYTE	**0x4A**		
Len	BYTE	1…129 (Number of Words)		
Parameters	BYTE	Length (max. 255 bytes)		
	BYTE	Toggle LastSegment	[Bit 0] [Bit 1]	Toggle Bit Last Data Segment
	BYTE [0…255]	Data Bytes Write		

Response Frame				
OpCode	BYTE	**0x00**		
Len	BYTE	3		
Parameters	DWORD	ErrorCode (➔page 10-70)		
	BYTE	Length written (max. 255 bytes)		
	BYTE	Toggle	[Bit 0]	Toggle Bit

10.1.2.3 NMT Service

«SendNMTService»

Send a NMT service to, for example, change NMT state or reset the device.

Request Frame		
OpCode	BYTE	**0x4B**
Len	BYTE	2
Parameters	WORD	Network ID
	BYTE	Node ID
	BYTE	Command Specifier

Response Frame		
OpCode	BYTE	**0x00**
Len	BYTE	2
Parameters	DWORD	ErrorCode (➔page 10-70)

10.1.3 General CAN Commands

«SendCANFrame»

Send a general CAN Frame to a CAN network specified by the Network ID.

Request Frame		
OpCode	BYTE	**0x50**
Len	BYTE	7
Parameters	WORD	Network ID
	WORD	Identifier
	WORD	Length
	BYTE	Data

Response Frame		
OpCode	BYTE	**0x00**
Len	BYTE	2
Parameters	DWORD	ErrorCode (➔page 10-70)

«RequestCANFrame»

Request a PDO/Guarding CAN Frame from a CAN network specified by the Network ID using Remote Transmit Request (RTR).

Request Frame		
OpCode	BYTE	**0x51**
Len	BYTE	3
Parameters	WORD	Network ID
	WORD	Identifier
	WORD	Length

Response Frame		
OpCode	BYTE	**0x00**
Len	BYTE	2
Parameters	DWORD	ErrorCode (➔page 10-70)
	BYTE [8]	Data

«SendLSSFrame»

Send a LSS master message to a CAN network specified by the Network ID.

Request Frame		
OpCode	BYTE	**0x54**
Len	BYTE	5
Parameters	WORD	Network ID
	BYTE [8]	Data

Response Frame		
OpCode	BYTE	**0x00**
Len	BYTE	2
Parameters	DWORD	ErrorCode (➔page 10-70)

«ReadLSSFrame»

Read a LSS slave message from a CAN network specified by Network ID.

Request Frame		
OpCode	BYTE	**0x55**
Len	BYTE	2
Parameters	WORD	Network ID
	BYTE [8]	Timeout

Response Frame		
OpCode	BYTE	**0x00**
Len	BYTE	2
Parameters	DWORD	ErrorCode (➔page 10-70)
	BYTE [8]	Data

10.2 USB & RS232 Communication

The present section describes the communication protocol between the Supervisor and the EPOS2 P Positioning Controller using USB or RS232 interface.

10.2.1 Data Link Layer

10.2.1.1 Flow Control

The EPOS2 P Positioning Controllers always communicates as a slave. A frame is only sent as an answer to a request. All EPOS2 P commands send an answer. The master always must start the communication by sending a packet structure.

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

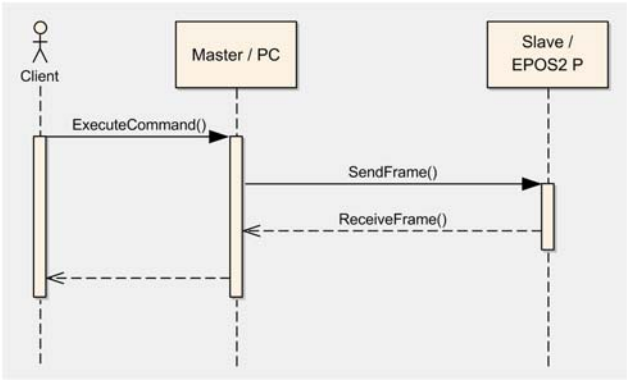


Figure 10-29 Protocol Sequence

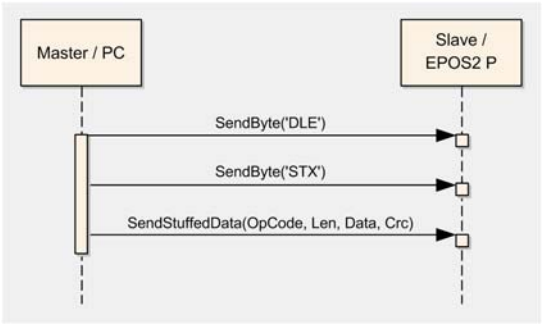


Figure 10-30 Sending a Data Frame to EPOS2 P

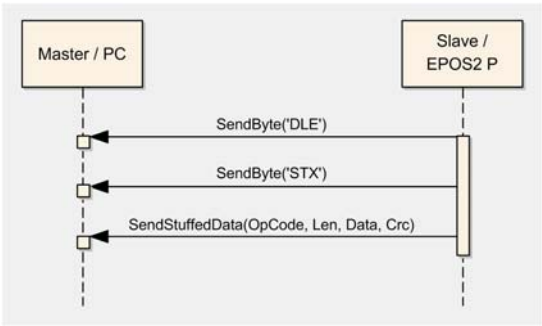


Figure 10-31 Receiving a Response Data Frame from EPOS2 P

10.2.1.2 Frame Structure

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

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

"DLE" (8-bit)	"STX" (8-bit)	OpCode (8-bit)	Len (8-bit)	Data[0] (16-bit)	…	Data[Len-1] (16-bit)	CRC (16-bit)
SYNC		HEADER		DATA			CRC

Figure 10-32 Frame Structure

SYNC The first two bytes are used for frame synchronization.

"DLE" Starting frame character "DLE" (Data Link Escape) = 0x90

"STX" Starting frame character "STX" (Start of Text) = 0x02

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

OpCode Operation command to be sent to the slave. For details on the command set ➔chapter "10.1 Command Reference (USB & RS232)" on page 10-41.

Len represents the number of words (16-bit value) in the data fields [0…143].

DATA The data field contains the parameters of the message. The low byte of the word is transmitted first.

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

CRC The 16-bit CRC checksum using the algorithm CRC-CCITT.
The CRC calculation includes all bytes of the frame except the synchronization bytes, the data bytes must be calculated as a word.
First, you will need to shift to the data word's high byte.
This represents the opposite way as you transmit the data word.
For calculation, the 16-bit generator polynomial "$x^{16}+x^{12}+x^5+1$" is used.

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

Remark
The CRC is calculated before stuffing the data. The elements "OpCode" to "Data[Len-1]" are included in CRC calculation. The synchronization elements "DLE" and "STX" are not included.

10.2.1.3 Error Control

Acknowledge

As a reaction to a bad OpCode or CRC value, the slave sends a frame containing the corresponding error code.

CRC Calculation

Packet M(x):	WORD DataArray[n]
Generator Polynom G(x):	10001000000100001 (= $x^{16}+x^{12}+x^5+x^0$)
DataArray[0]:	HighByte(Len) + LowByte(**OpCode**)
DataArray[1]:	**Data[0]**
DataArray[2]:	**Data[1]**
…	…
DataArray[n-1]:	**0x0000 (ZeroWord)**

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

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

Figure 10-33 CRC Calculation

10.2.1.4 Character Stuffing

The sequence “DLE” and “STX” are reserved for frame start synchronization. If the character “DLE” appears at a position between “OpCode” and “CRC” and is not a starting character, the character must be doubled (character stuffing). Otherwise, the protocol begins to synchronize for a new frame. The character “STX” needs not to be doubled.

Examples:

Sending Data	0x21, **0x90**, 0x45
Stuffed Data	0x21, **0x90**, **0x90**, 0x45

Sending Data	0x21, **0x90, 0x02,** 0x45
Stuffed Data	0x21, **0x90**, **0x90**, **0x02**, 0x45

Sending Data	0x21, **0x90**, **0x90**, 0x45
Stuffed Data	0x21, **0x90**, **0x90**, **0x90**, **0x90**, 0x45

Important!
Character stuffing is used for all bytes in the frame except the starting characters.

10.2.1.5 Transmission Byte Order

The unit of data memory in EPOS2 P is a word (16-bit value). To send and receive a word (16-bit) via the serial port, 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 following order: byte0 (LSB), byte1 (MSB).
A long word will be transmitted in following order: byte0 (LSB), byte1, byte2, byte3 (MSB).

10.2.1.6 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.

Object	Index	SubIndex	Default
RS232 Frame Timeout	0x2005	0x00	500 [ms]
USB Frame Timeout	0x2006	0x00	500 [ms]

Table 10-29 Communication – Timeout Handling

Remark
To cover special requirements, the timeout may be changed by writing to the Object Dictionary!

10.2.2 EPOS2 P State Machine

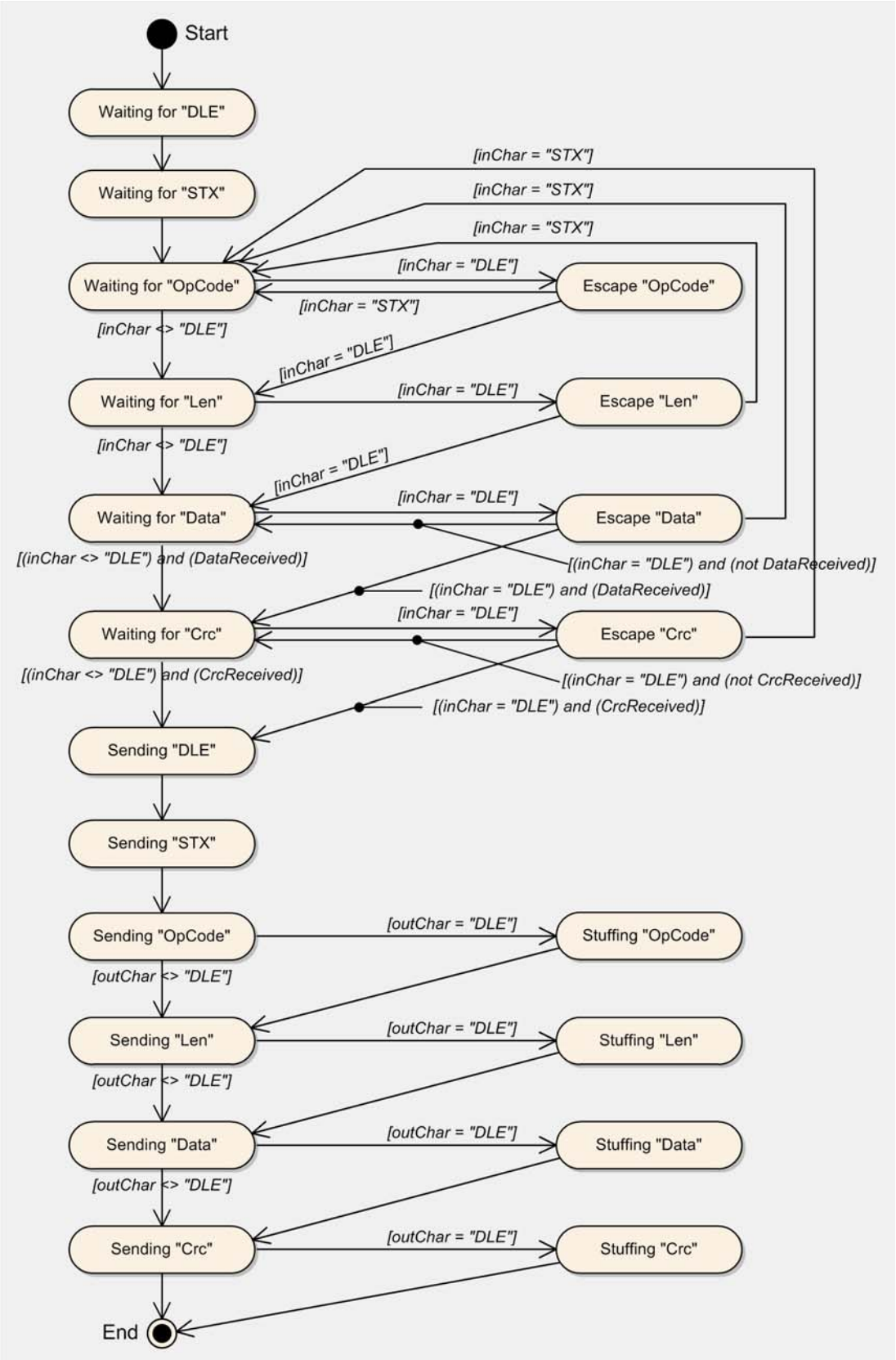


Figure 10-34 EPOS2 P State Machine

10.2.3 Physical Layer

10.2.3.1 USB

Electrical Standard

maxon EPOS2 P drives' USB interface follows the «Universal Serial Bus Specification Revision 2.0». You may wish to download the file from the Internet (for URL ➔"Sources for additional Information" on page 1-6), full details are described in chapter "7.3 Physical Layer".

10.2.3.2 RS232

Electrical Standard

The EPOS2 P communication protocol uses the RS232 standard to transmit data over a 3-wire cable (signals TxD, RxD and GND).

The RS232 standard can only be used for point-to-point communication between a master and a single EPOS2 P slave. It uses negative, bipolar logic with a negative voltage signal representing a logic "1", and positive voltage representing a logic "0". Voltages of −3…−25 V with respect to signal ground (GND) are considered logic "1", whereas voltages of +3…25 V are considered logic "0".

Medium

For the physical connection, a 3-wire cable will be required. We recommend to install a shielded and twisted pair cable in order to achieve good performance, even in an electrically noisy environment. Depending on the bit rate used, the cable length can range from 3…15 meters. However, we do not recommend to use RS232 cables longer than 5 meters.

10.3 CANopen Communication

10.3.1 General Information

maxon EPOS2 P drives' CAN interface follows the CANopen standard CiA 301 «Communication Profile for Industrial Systems».

10.3.2 Documentation

For further information on CAN/CANopen as well as respective specifications ➔listed references in chapter "1.5 Sources for additional Information" on page 1-6.

10.3.3 Notations, Abbreviations and Terms used

Notation	Description	Format
nnnnb	Numbers followed by "b".	binary
nnnnh	Numbers followed by "h".	hexadecimal
nnnn	All other numbers.	decimal

Table 10-30 CAN Communication – Notations

Abbreviation	Description
CAN	CAN Application Layer
CMS	CAN Message Specification
COB	Communication Object (CAN Message) – a unit of transportation in a CAN message network. Data must be sent across a network inside a COB.
COB-ID	COB Identifier – identifies a COB uniquely in a network and determines the priority of that COB in the MAC sublayer.
EDS	Electronic Data Sheet – a standard form of all CAN objects supported by a device. Used by external CAN configurators.
ID	Identifier – the name by which a CAN device is addressed.
MAC	Medium Access Control – one of the sublayers of the Data Link Layer in the CAN Reference Model. Controls the medium permitted to send a message.
OD	Object Dictionary – the full set of objects supported by the node. Represents the interface between application and communication (➔term "Object" on page 10-53).
PDO	Process Data Object – object for data exchange between several devices.
PLC	Programmable Controller – can serve as a CAN Master for the EPOS2.
RO	Read Only
RW	Read Write
SDO	Service Data Object – peer-to-peer communication with access to the device's Object Directory.
WO	Write Only

Table 10-31 CAN Communication – Abbreviations

Term	Description
CAN Client or **CAN Master**	A host (typically a PC or other control equipment) supervising the nodes of a network.
CAN Server or **CAN Slave**	A node in the CAN network that can provide service under the CAN Master's control.
Object	A CAN message with meaningful functionality and/or data. Objects are referenced according to addresses in the Object Dictionary.
Receive	"received" data is being sent from the control equipment to the EPOS2.
Transmit	"transmitted" data is being sent from the EPOS2 to the other equipment.

Table 10-32 CAN Communication – Terms

10.3.4 CANopen Basics

Subsequently described are the CANopen communication features most relevant to the maxon motor's EPOS2 P Programmable Positioning Controllerss. 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.

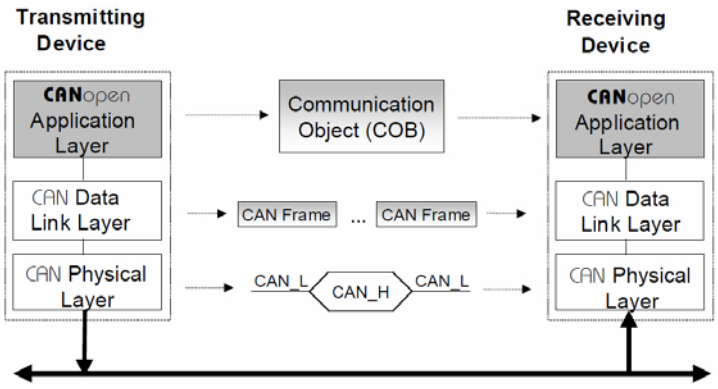


Figure 10-35 CAN Communication – Protocol Layer Interactions

10.3.4.1 Physical Layer

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.

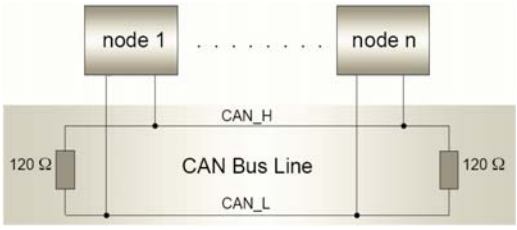


Figure 10-36 CAN Communication – ISO 11898 Basic Network Setup

10.3.4.2 Data Link Layer

The CAN data link layer is also standardized in ISO 11898. Its services are implemented in the Logical Link Control (LLC) and Medium Access Control (MAC) sublayers of a CAN controller.

- The LLC provides acceptance filtering, overload notification and recovery management.
- The MAC is responsible for data encapsulation (decapsulation), frame coding (stuffing/destuffing), medium access management, error detection, error signaling, acknowledgement, and serialization (deserialization).

A Data Frame is produced by a CAN node when the node intends to transmit data or if this is requested by another node. Within one frame, up to 8 byte data can be transported.

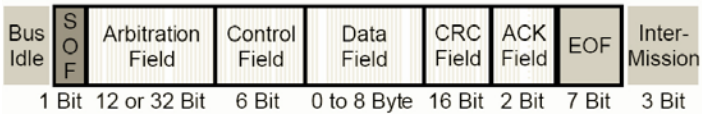


Figure 10-37 CAN Communication – CAN Data Frame

- 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 Acknowledgement (ACK) field, the transmitting node sends out a recessive bit. Any node that has received an error-free frame acknowledges the correct reception of the frame by 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.

With EPOS2 P, only the Standard Frame Format is supported.

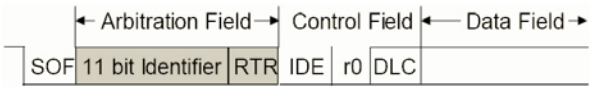


Figure 10-38 CAN Communication – Standard Frame Format

- 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.

10.3.5 CANopen Application Layer

10.3.5.1 Object Dictionary

The most significant part of a CANopen device is the Object Dictionary. It is essentially a grouping of objects accessible via the network in an ordered, predefined fashion. Each object within the dictionary is addressed using a 16-bit index and a 8-bit subindex. The overall layout of the standard Object Dictionary conforms to other industrial field bus concepts.

Index	Variable accessed
0000h	Reserved
0001h-009Fh	Data types (not supported on EPOS2)
00A0h-0FFFh	Reserved
1000h-1FFFh	Communication Profile Area (CiA 301)
2000h-5FFFh	Manufacturer-specific Profile Area (maxon motor
6000h-9FFFh	Standardized Device Area (CiA 402)
A000h-FFFFh	Reserved

Table 10-33 CAN Communication – Object Dictionary Layout

A 16-bit index 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 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.

An example: A receive PDO, the data structure at index 1400h defines the communication parameters for that module. This structure contains fields or the COB-ID and the transmission type. The subindex concept can be used to access these individual fields as shown below.

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

Table 10-34 CAN Communication – Object Dictionary Entry

10.3.5.2 Communication Objects

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

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

Communication Objects	
Process Data Objects (PDO)	
Service Data Objects (SDO)	
Special Function Objects Synchronization Objects (SYNC)	Time Stamp Objects (not used on EPOS2)
	Emergency Objects (EMCY)
Network Management Objects	NMT Message
	Node Guarding Object

Table 10-35 CAN Communication – Communication Objects

10.3.5.3 Predefined Communication Objects

PDO Object

PDO communication can be described by the producer/consumer model. Process data can be transmitted from one device (producer) to one another device (consumer) or to 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.

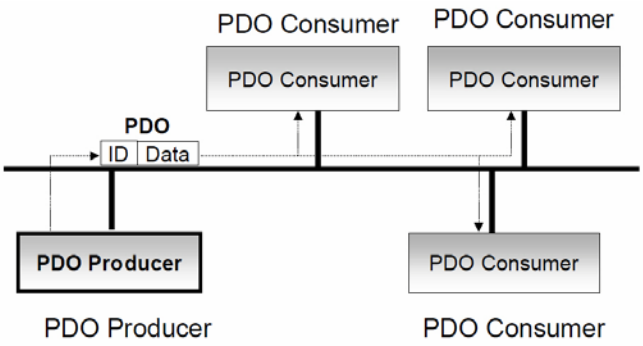


Figure 10-39 CAN Communication – Process Data Object

There are two PDO services:

- The Write PDO is mapped to a single CAN Data frame.
- The Read PDO is mapped to CAN Remote Frame, which will be responded by the corresponding CAN Data Frame.

Read PDOs are optional and depend on the device capability. The complete data field of up to 8 byte may contain process data. Number and length of a device's PDOs are application-specific and must be specified in the device profile.

The number of supported PDOs is accessible at the Object Dictionary's index 1004h. The PDOs correspond to entries in the Object Dictionary and serve as interface to application objects. Application objects' data type and mapping into a PDO is determined by a corresponding default PDO mapping structure within the Object Dictionary. This structure is defined in the entries "1600h" (for the first R_PDO) and "1A00h" (for the first T_PDO). In a CANopen network, up to 512 T_PDOs and 512 R_PDOs may be used.

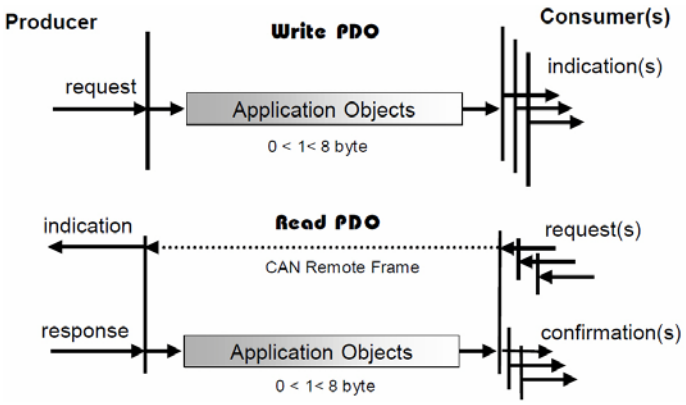


Figure 10-40 CAN Communication – PDO Protocol

The CANopen communication profile distinguishes three message triggering modes:

a) Message transmission is triggered by the occurrence of an object-specific event specified in the device profile.

b) The transmission of asynchronous PDOs may be initiated upon receipt of a remote request initiated by another device.

c) Synchronous PDOs are triggered by the expiration of a specified transmission period synchronized by the reception of the SYNC object.

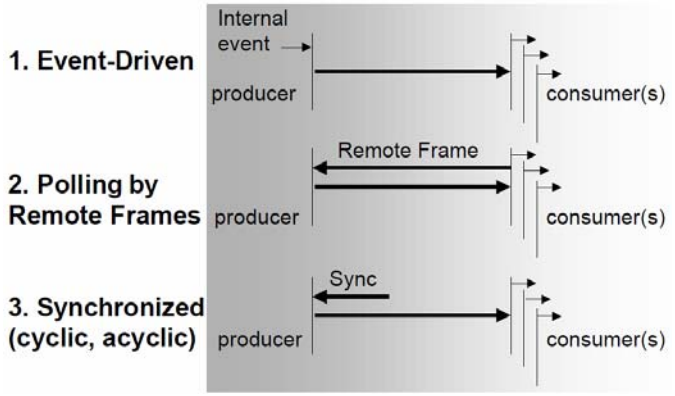


Figure 10-41 CAN Communication – PDO Communication Modes

SDO Object

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.

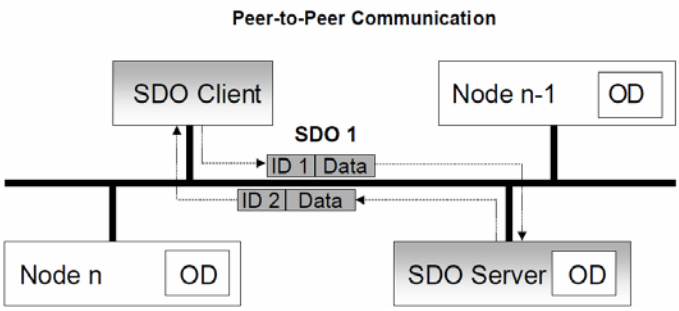


Figure 10-42 CAN Communication – Service Data Object

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

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

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

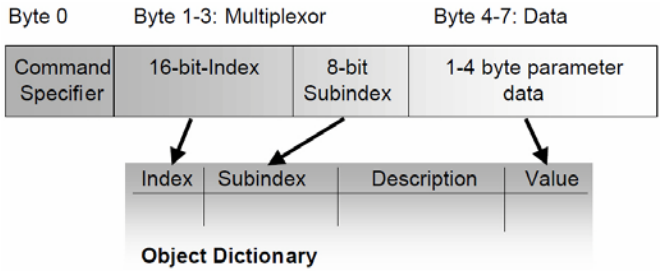


Figure 10-43 CAN Communication – Object Dictionary Access

SDO Network Indication Protocol

The network indication protocol manages the forwarding of remote communication services. Each SDO remote download or upload services starts with the SDO network indication service as defined in the figure below.

The service is confirmed. The remote result parameter indicates the success of the request. In case of a failure, an SDO abort transfer request is initiated. In the case of a success, the SDO expedited or normal or block service is performed. The SDO service is transparent to the device. The SDO network indication protocol is aborted by the client or the server using the abort SDO transfer message as defined in CANopen standard CiA 301.

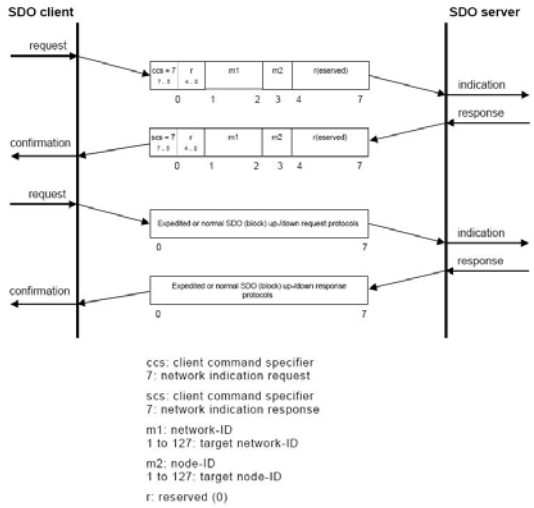


Figure 10-44 SDO Network Indication Protocol

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".

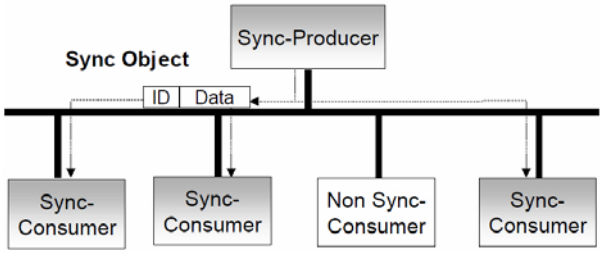


Figure 10-45 CAN Communication – Synchronization Object

Synchronous transmission of a PDO means that the transmission is fixed in time with respect to the transmission of the SYNC Object. The synchronous PDO is transmitted within a given time window "synchronous window length" with respect to the SYNC transmission and, at 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.

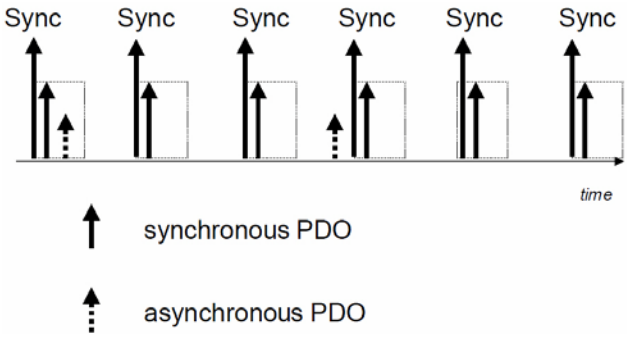


Figure 10-46 CAN Communication – Synchronous PDO

EMERGENCY Object

Emergency messages are triggered by the occurrence of a device internal fatal error. They are transmitted by the concerned device to the other devices with high priority, thus making them suitable for interrupt type error alerts.

An Emergency Telegram may be sent only once per “error event”, i.e. the emergency messages must not be repeated. As long as no new errors occur on a Enter Pre-Operational device, no further emergency message must be sent. The error register as well as additional, device-specific information are specified in the device profiles by means of emergency error codes defined as to CANopen Communication Profile.

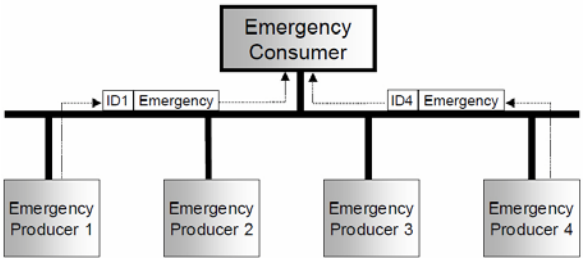


Figure 10-47 CAN Communication – Emergency Service

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.

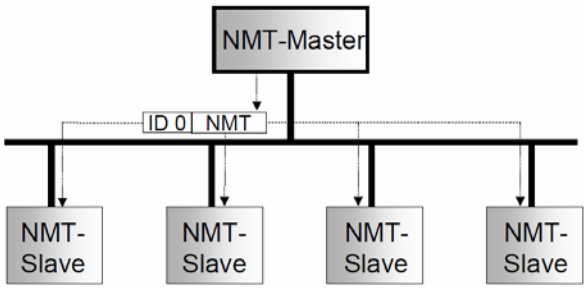


Figure 10-48 CAN Communication – Network Management (NMT)

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

In "Pre-Operational" state, the node may be configured and parameterized via SDO (e.g. using a configuration tool), PDO communication is not permitted. The NMT Master may switch from "Pre-Operational" to "Operational", and vice versa.

In "Operational" state, PDO transfer is permitted. By switching a device into "Stopped" state it will be forced to stop PDO and SDO communication. 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).

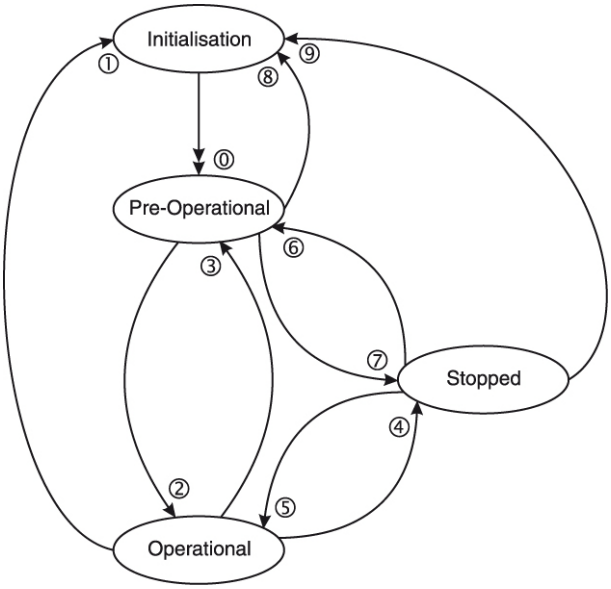


Figure 10-49 CAN Communication – NMT Slave State Diagram

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

Remarks:

*1) Command may be sent with Network Management (NMT) Protocol.
*2) This Transition is generated automatically by the EPOS2 P after initialization is completed. After initialization a Boot-Up message is send.
*3) Remote flag Bit 9 of the Statusword.

Service *1	Transition	NMT State after Command	Remote *3	Functionality
–*2	0	Pre-Operational	FALSE	Communication: – Service Data Objects (SDO) Protocol – Emergency Objects – Network Management (NMT) Protocol
Enter Pre-Operational	3, 6	Pre-Operational	FALSE	
Reset Communication	1, 8, 9	Initialization (Pre-Operational)	FALSE	Calculates SDO COB-IDs. Setup Dynamic PDO-Mapping and calculates PDO COB-IDs. Communication: – While initialization is active, no communication is supported. – Upon completion, a boot-up message will be sent to the CAN Bus.
Reset Node	1, 8, 9	Initialization (Pre-Operational)	FALSE	Generates a general reset of EPOS2 P software having same effect as turning off and on the supply voltage. Not saved parameters will be overwritten with values saved to the EEPROM using «Save all Parameters».
Start Remote Node	2, 5	Operational	TRUE	Communication: – Service Data Objects (SDO) Protocol – Process Data Objects (PDO) Protocol – Emergency Objects – Network Management (NMT) Protocol
Stop Remote Node	4, 7	Stopped	FALSE	Communication: – Network Management (NMT) Protocol – Layer setting services (LSS) – Lifeguarding (Heartbeating)

Table 10-36 CAN Communication – NMT Slave (Commands, Transitions and States)

The communication object possesses the identifier (=0) and consists of two bytes. The Node ID defines the destination of the message. If zero, the protocol addresses all NMT Slaves.

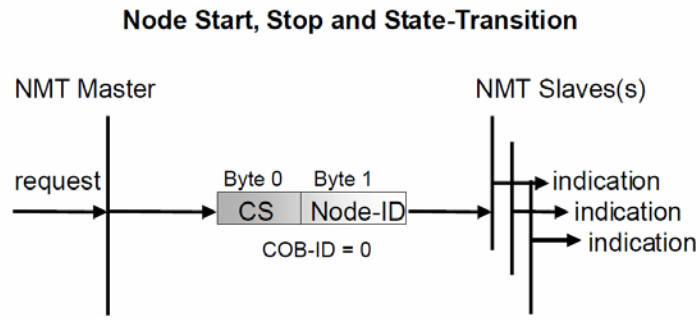


Figure 10-50 CAN Communication – NMT Object

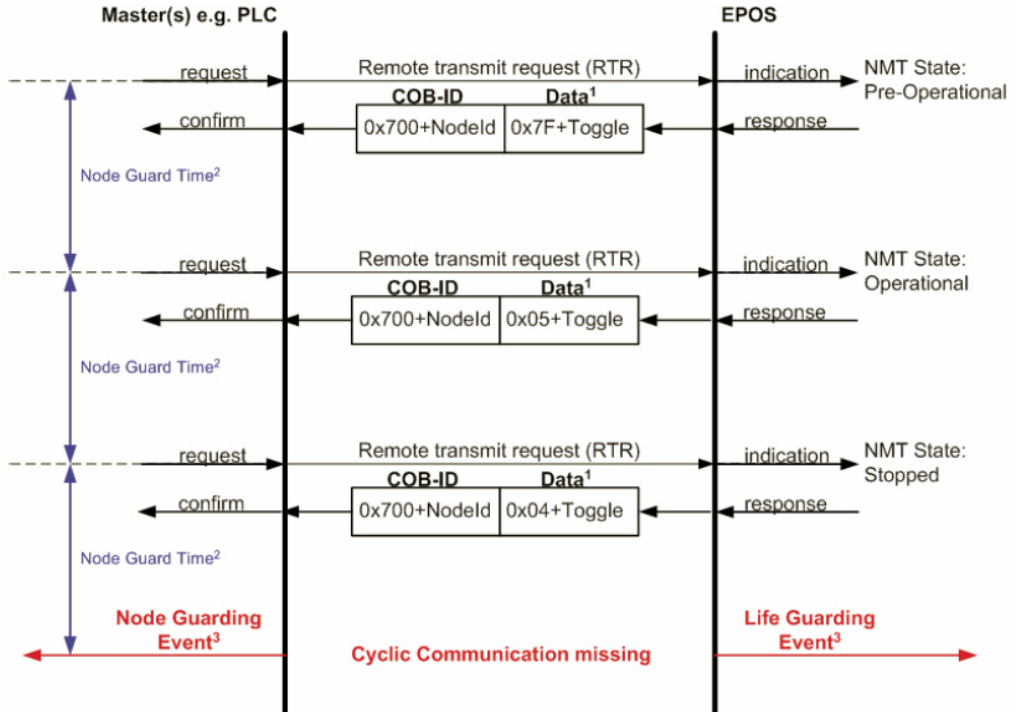
Protocol	COB-ID	CS (Byte 0)	Node ID (Byte 1)	Functionality
Enter Pre-Operational	0	0x80	0 (all)	All CANopen nodes (EPOS2 P devices) will enter NMT State “Pre-Operational”.
	0	0x80	n	The CANopen node (EPOS2 P device) with Node ID “n” will enter NMT State “Pre-Operational”.
Reset Communication	0	0x82	0 (all)	All CANopen nodes (EPOS2 P devices) will reset the communication.
	0	0x82	n	The CANopen node (EPOS2 P device) with Node ID “n” will reset the communication.
Reset Node	0	0x81	0 (all)	All CANopen nodes (EPOS2 P devices) will reset.
	0	0x81	n	The CANopen node (EPOS2 P device) with Node ID “n” will reset.
StartRemoteNode	0	0x01	0 (all)	All CANopen nodes (EPOS2 P devices) will enter NMT State “Operational”.
	0	0x01	n	The CANopen node (EPOS2 P device) with Node ID “n” will enter NMT State “Operational”.
COB-ID	0	0x02	0 (all)	All CANopen nodes (EPOS2 P devices) will enter NMT State “Stopped”.
	0	0x02	n	The CANopen node (EPOS2 P device) with Node ID “n” will enter NMT State “Stopped”.

Table 10-37 CAN Communication – NMT Protocols

Node Guarding Protocol

Used to detect absent devices that do not transmit PDOs regularly (e.g. due of “bus off”). The NMT Master can manage “Node Guarding”, a database where, among other information, expected states of all connected devices are recorded. With cyclic Node Guarding, the NMT Master regularly polls its NMT Slaves. To detect the absence of the NMT Master, the slaves test internally, whether Node Guarding is taking place in the defined time interval (Life Guarding).

Node Guarding is initiated by the NMT Master (in “Pre-Operational” state of the slaves) by transmitting a Remote Frame. Node Guarding is also activated if “Stopped” state is active.



Legend: 1) Data Field / 2) Node Guard Time / 3) Node/Life Guarding Event

Figure 10-51 CAN Communication – Node Guarding Protocol (Timing Diagram)

A) Data Field

Holds the NMT State. Upon receipt of a node guard answer, bit 8 toggles between 0x00 and 0x80. Thus, the data field supports the following values:

Value	Toggle	EPOS2 NMT State
0x04	not set	Stopped
0x84	set	Stopped
0x05	not set	Operational
0x85	set	Operational
0x7F	not set	Pre-Operational
0xFF	set	Pre-Operational

Table 10-38 CAN Communication – Node Guarding Protocol (Data Field)

B) Node Guard Time

Is calculated as follows: $NodeGuardTime = GuardTime \cdot LifeTimeFactor$

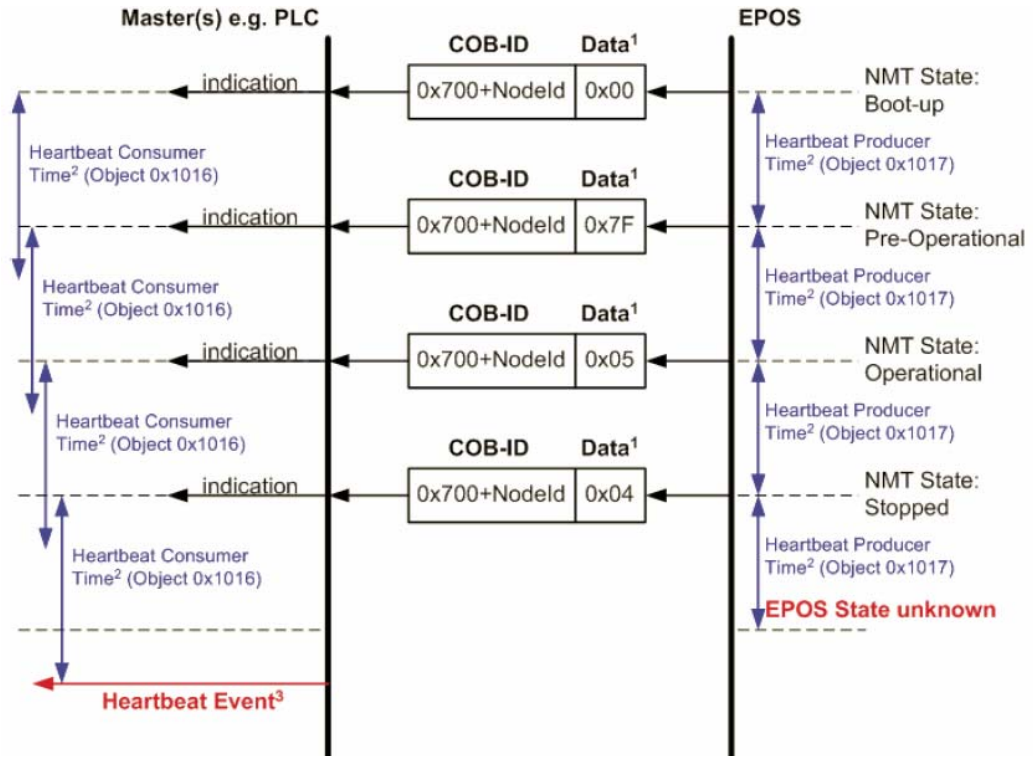
C) Node / Life Guarding Event

In case EPOS2 P misses the Remote Transmit Request (RTR), it will change it’s device state to “Error” (Node Guarding Error).

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

Heartbeat Protocol

The Heartbeat Protocol has a higher priority than the Node Guarding Protocol. If both are enabled, only the Heartbeat Protocol is supported. The EPOS2 P transmits a cyclic heartbeat message if the Heartbeat Protocol is enabled (Heartbeat Producer Time 0 = Disabled / greater than 0 = enabled). The Heartbeat Consumer guards receipt of the Heartbeat within the Heartbeat Consumer Time. If the Heartbeat Producer Time is configured in EPOS2 P, it will start immediately with the Heartbeat Protocol.



Legend: 1) Data Field / 2) Heartbeat Producer and Heartbeat Consumer Time / 3) Hearbeat Event

Figure 10-52 CAN Communication – Heartbeat Protocol (Timing Diagram)

A) Data Field

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

Value	EPOS2 NMT State
0x00	Bootup
0x04	Stopped
0x05	Operational
0x7F	Pre-Operational

Table 10-39 CAN Communication – Heartbeat Protocol (Data Field)

B) Heartbeat Producer Time and Heartbeat Consumer Time

The Heartbeat Consumer Time must be longer than the Heartbeat Producer Time because of generation, sending and indication time ($HeartbeatConsumerTime \geq HeartbeatProducerTime + 5ms$). Each indication of the Master resets the Heartbeat Consumer Time.

C) Heartbeat Event

If EPOS2 is in an unknown state (e.g. supply voltage failure), the Heartbeat Protocol cannot be sent to the Master. The Master will recognize this event upon elapsed Heartbeat Consumer Time and will generate a Heartbeat Event.

10.3.6 Identifier Allocation Scheme

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

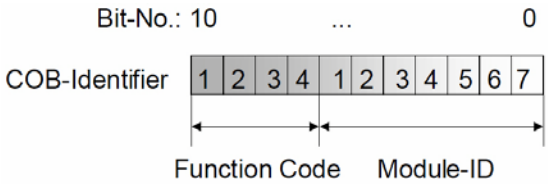


Figure 10-53 CAN Communication – Default Identifier Allocation Scheme

This ID allocation scheme allows peer-to-peer communication between a single master device and up to 127 slave devices. It also supports broadcasting of non-confirmed NMT Services, SYNC and Node Guarding.

The predefined master/slave connection set supports…

- one emergency object,
- one SDO,
- four Receive PDOs and three Transmit PDOs and the
- node guarding object.

Object	Function Code (binary)	Resulting COB-ID		Communication Parameter at Index
NMT	0000	0		–
SYNC	0001	128	(0080h)	1005h
EMERGENCY		129…255	(0081h-00FFh)	1014h
PDO1 (tx)	0011	385…511	(0181h-01FFh)	1800h
PDO1 (rx)	0100	513…639	(0201h-027Fh)	1400h
PDO2 (tx)	0101	641…8767	(0281h-02FFh)	1801h
PDO2 (rx)	0110	769…895	(0301h-037Fh)	1401h
PDO3 (tx)	0111	897…1023	(0381h-03FFh)	1802h
PDO3 (rx)	1000	1025…1151	(0401h-047Fh)	1402h
PDO4 (tx)	1001	1153…1279	(0481h-04FFh)	1803h
PDO4 (rx)	1010	1281…1407	(0501h-057Fh)	1403h
SDO1 (tx)	1011	1409…1535	(0581h-05FFh)	1200h
SDO1 (rx)	1100	1537…1663	(0601h-067Fh)	1200h

Table 10-40 CAN Communication – Objects of the Default Connection Set

10.4 Gateway Communication

10.4.1 USB & RS232 to CAN Gateway

Using the gateway functionality, the Supervisor can access all other EPOS2 devices connected to the CAN Bus via the gateway device's USB port or RS232 interface. Even other CANopen devices (I/O modules) supporting the CANopen standard, CiA 301 may be accessed.

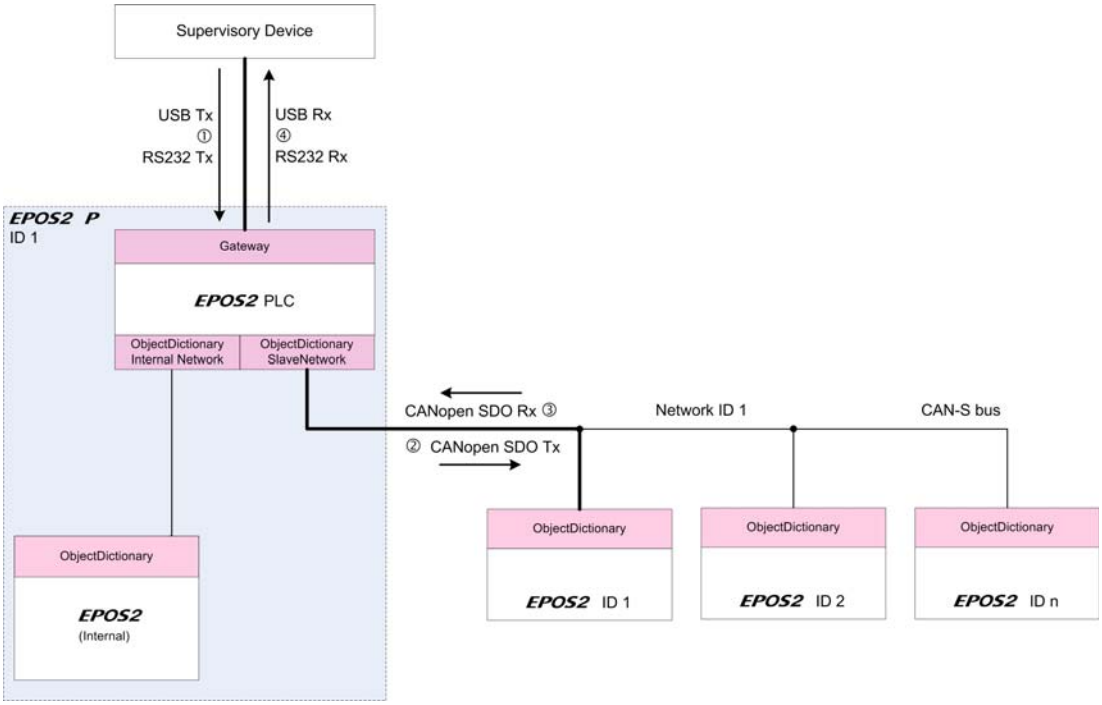


Figure 10-54 USB/RS232 to CAN Gateway Communication – Structure

Communication data are exchanged between Supervisor and the gateway using a maxon-specific USB/RS232 protocol.

Data between the gateway and the addressed device are exchanged using the CANopen SDO protocol according to the CiA 301.

Step	Protocol	Sender → Receiver	Description	
1	USB or RS232	CANopen Supervisor ↓ EPOS2 PLC (Gateway)	Command including Network ID and Node ID is sent to the device (EPOS2 P) working as a gateway. The gateway decides whether to execute the command or to translate and forward it to the CAN Bus dedicated by Network ID.	
			Criteria: Network ID = 0 & Node ID = 0 (Gateway) Network ID = 0 & Node ID = DIP switch else (Network ID not 0)	 → Execute → Execute → Forward to CAN
2	CANopen [SDO]	EPOS2 PLC (Gateway) ↓ EPOS2 Network ID 1, Node ID 1	The gateway is forwarding the command to the CAN network. The USB/RS232 command is translated to a CANopen SDO service.	
3	CANopen [SDO]	EPOS2 Network ID 1, Node ID 1 ↓ EPOS2 PLC (Gateway)	The EPOS2 Network ID 1, Node ID 1 is executing the command and sending the corresponding CAN frame back to the gateway.	
4	USB or RS232	EPOS2 PLC (Gateway) ↓ Supervisor	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.	

Table 10-41 USB/RS232 to CAN Gateway Communication – Data Exchange

10.4.2 CAN to CAN Gateway

Using the gateway functionality, the Supervisor can access all other EPOS2 devices connected to the CAN Bus via the gateway device's CAN-M interface. Even other CANopen devices (I/O modules) supporting the CANopen standard, CiA 301 may be accessed.

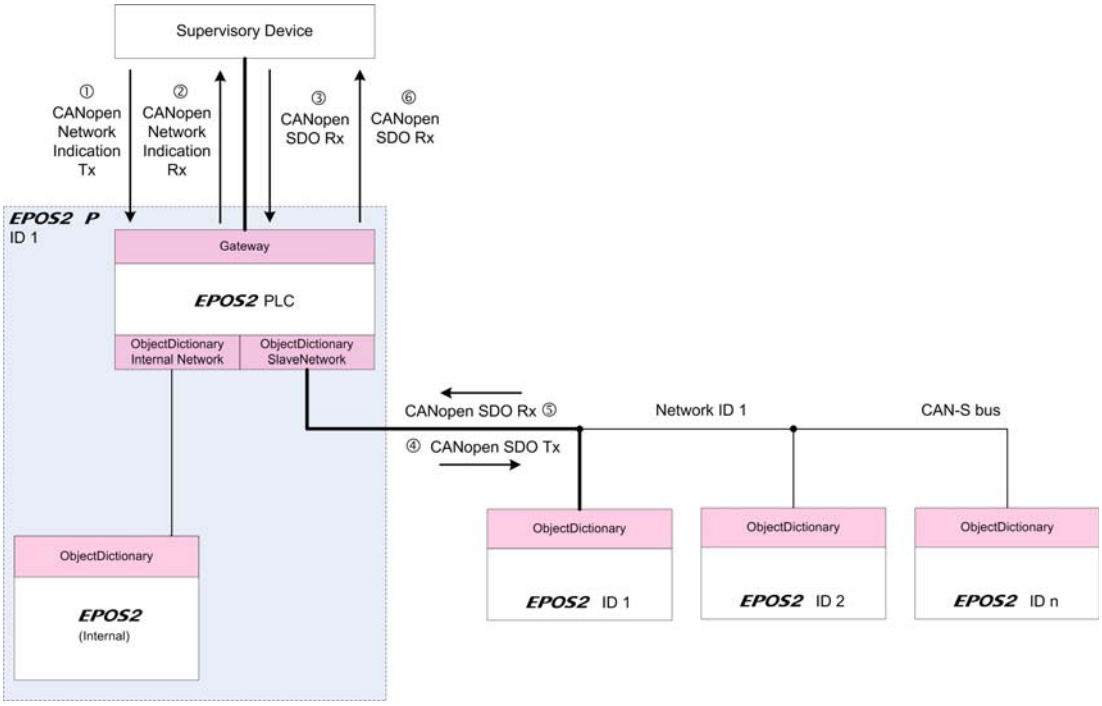


Figure 10-55 CAN to CAN Gateway Communication – Structure

Step	Protocol	Sender → Receiver	Description
1	CANopen Network	CANopen Supervisor ↓ EPOS2 PLC (Gateway)	The Supervisor is sending the CAN frame corresponding to the SDO network indication protocol. It includes the target Network ID and the target Node ID.
2	CANopen Network Identification	EPOS2 PLC (Gateway) ↓ CANopen Supervisor	The gateway is sending the network identification response back to the Supervisor.
3	CANopen [SDO]	CANopen Supervisor ↓ EPOS2 PLC (Gateway)	The Supervisor is sending the SDO CAN frame containing the data to be sent to the target device.
4	CANopen [SDO]	EPOS2 PLC (Gateway) ↓ EPOS2 Network ID 1, Node ID 1	The gateway is forwarding the command to the CAN network.
5	CANopen [SDO]	EPOS2 Network ID 1, Node ID 1 ↓ EPOS2 PLC (Gateway)	The EPOS2 Network ID 1, Node ID 1 is executing the command and sending the corresponding CAN frame back to the gateway.
6	CANopen [SDO]	EPOS2 PLC (Gateway) ↓ CANopen Supervisor	The gateway is receiving the CAN frame corresponding to the SDO service. This CAN frame is sent back to the CANopen Supervisor.

Table 10-42 CAN to CAN Gateway Communication – Data Exchange

10.5 Error Code Definition

10.5.1 CANopen-specific Error Codes

Following error codes are defined by CANopen standard CiA 301 «Communication Profile for Industrial Systems»:

Error Code	Name	Cause
0x0000 0000	No Communication Error	RS232 communication successful
0x0503 0000	Toggle Error	Toggle bit not alternated
0x0504 0000	SDO Time Out	SDO protocol timed out
0x0504 0001	Client / Server Specifier Error	Client / server command specifier not valid or unknown
0x0504 0004	CRC Error	CRC
0x0504 0005	Out of Memory Error	Out of memory
0x0601 0000	Access Error	Unsupported access to an object
0x0601 0001	Write Only Error	Read command to a write only object
0x0601 0002	Read Only Error	Write command to a read only object
0x0602 0000	Object does not exist Error	Object does not exist in Object Directory. Last read or write command had wrong object Index or SubIndex
0x0604 0041	PDO mapping Error	Object cannot be mapped to the PDO
0x0604 0042	PDO Length Error	Number and length of objects to be mapped would exceed PDO length
0x0604 0043	General Parameter Error	General parameter incompatibility
0x0604 0047	General internal Incompatibility Error	General internal incompatibility in device
0x0606 0000	Hardware Error	Access failed due to hardware error
0x0607 0010	Service Parameter Error	Data type does not match, length or service parameter does not match
0x0607 0012	Service Parameter too long Error	Data type does not match, length of service parameter too high
0x0607 0013	Service Parameter too short Error	Data type does not match, length of service parameter too low
0x0609 0011	Object SubIndex Error	Last read or write command had wrong object SubIndex
0x0609 0030	Value Range Error	Value range of parameter exceeded
0x0609 0031	Value too high Error	Value of parameter written too high
0x0609 0032	Value too low Error	Value of parameter written too low
0x0609 0036	Maximum less Minimum Error	Maximum value is less than minimum value
0x0800 0000	General Error	General error
0x0800 0020	Transfer or Store Error	Data cannot be transferred/stored
0x0800 0021	Local Control Error	Data cannot be transferred/stored to application due to local control

Error Code	Name	Cause
0x0800 0022	Wrong Device State	Data cannot be transferred/stored to application due to present device state
0x0A00 0001	Network Id unknown	Network Id is unknown (does not exist in routing list)
0x0A00 0002	Node ID unknown	Node ID is unknown

Table 10-43 Communication Errors – CANopen-specific Error Codes

10.5.2 maxon-specific Error Codes

Following error codes are specific to maxon's EPOS2 P devices:

Error Code	Name	Cause
0x0F00 FFB9	CAN ID Error	Wrong CAN ID
0x0F00 FFBC	Service Mode Error	Device is not in service mode
0x0F00 FFBE	Password Error	Password is incorrect
0x0F00 FFBF	Illegal Command Error	RS232 command is illegal (does not exist)
0x0F00 FFC0	Wrong NMT State Error	Device is in wrong NMT state
0x0F01 0110	Program flashing error	For details ➔Flash Status Identification on page 8-37
0x0FFF FFF0	Communication Sequence Error	Error during function block execution
0x0FFF FFF1	Communication aborted	Communication has been aborted
0x0FFF FFF2	Communication Buffer Overflow	Communication buffer overflow
0x0FFF FFF9	Segmented Transfer Communication Error	Segmented transfer communication error
0x0FFF FFFA	Wrong Axis Number	The axis number was not within 0…32
0x0FFF FFFB	Wrong CAN Device	The can device number was not within 0…127
0x0FFF FFFC	Wrong CAN Port	CAN Port is not valid (not 1 or 2)
0x0FFF FFFD	Wrong Parameter	Internal function calling parameters wrong
0x0FFF FFFE	General Communication Error	General communication error
0x0FFF FFFF	Communication Timeout	Communication timeout occurred

Table 10-44 Communication Errors – maxon-specific Error Codes

••page intentionally left blank••

LIST OF FIGURES

LIST OF TABLES

INDEX

P

R

S

T

V

W

••page intentionally left blank••

maxon motor ag
Brünigstrasse 220
P.O.Box 263
CH-6072 Sachseln
Switzerland

Phone +41 41 666 15 00
Fax +41 41 666 16 50

www.maxonmotor.com