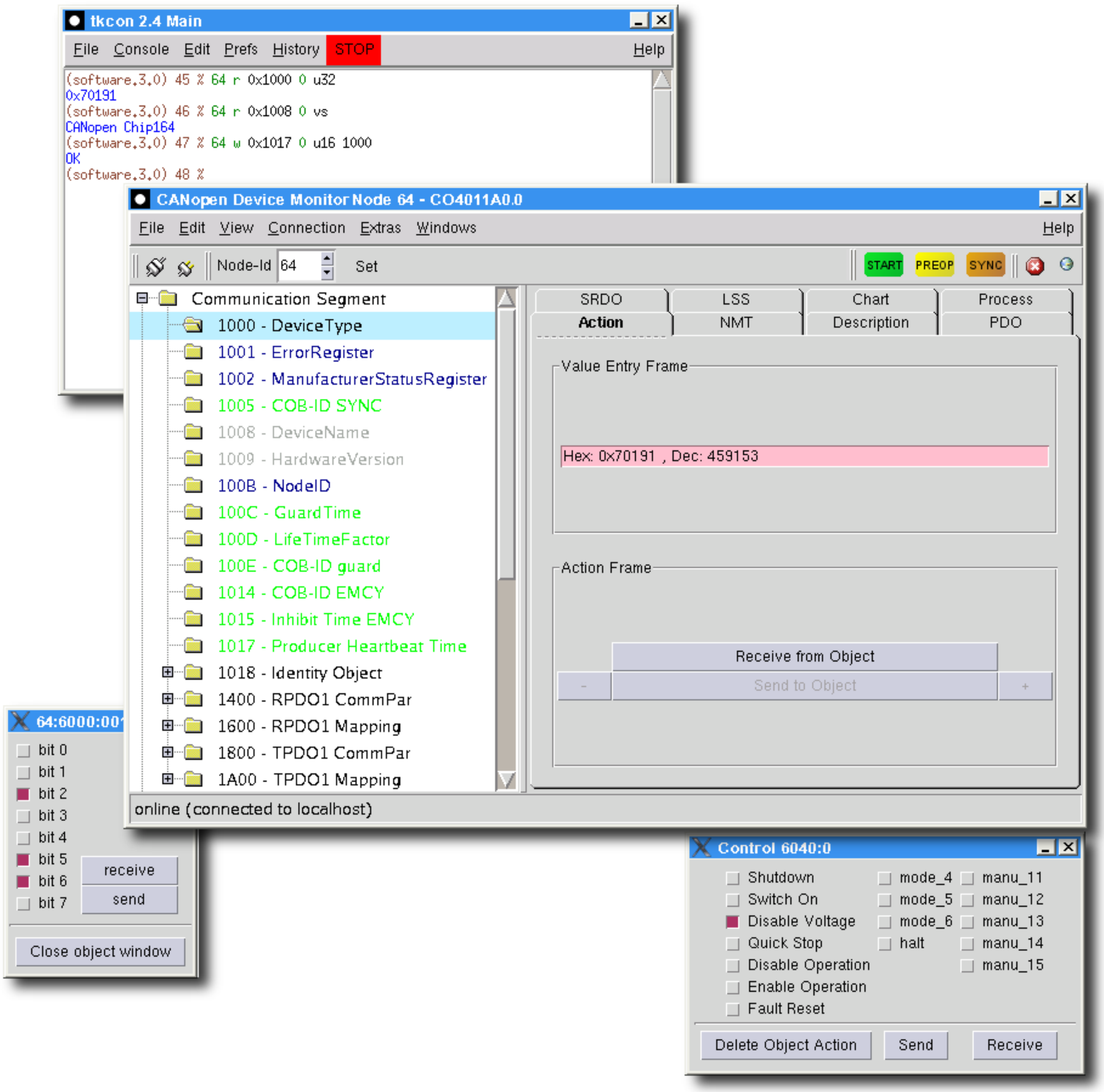
port

CANopen Device Monitor
User Manual



```
(software.3.0) 45 % 64 r 0x1000 0 u32
0x70191
(software.3.0) 46 % 64 r 0x1008 0 vs
CANopen Chip164
(software.3.0) 47 % 64 w 0x1017 0 u16 1000
OK
(software.3.0) 48 %
```

Hex: 0x70191 , Dec: 459153

online (connected to localhost)

© *port* GmbH, Halle 11.01.2013; CANopen Device Monitor Version 3.2.7

Disclaimer

All rights reserved

The programs, boards and documentations supplied by *port* GmbH are created with due diligence, checked carefully and tested on several applications.

Nevertheless, *port* GmbH can not take over no guarantee and no assume del credere liability that the program, the hardware board and the documentation are error-free respective are suitable to serve the special purpose.

In particular performance characteristics and technical data given in this document may not be constituted to be guaranteed product features in any legal sense.

For consequential damages, which are emerged on the strength of use the program and the hardware boards therefore, every legal responsibility or liability is excluded.

port has the right to modify the products described or their documentation at any time without prior warning, as long as these changes are made for reasons of reliability or technical improvement.

All rights of this documentation lie with *port*. The transfer of rights to third parties or duplication of this document in any form, whole or in part, is subject to written approval by *port*. Copies of this document may however be made exclusively for the use of the user and his engineers. The user is thereby responsible that third parties do not obtain access to these copies.

The soft- and hardware designations used are mostly registered and are subject to copyright.

CANopen®
is registered trademark, licensed by CiA - CAN in Automation e.V., Germany.

EtherCAT®
is registered trademark and patented technology, licensed by Beckhoff Automation GmbH, Germany.

We are thankful for hints of possible errors and may ask around for an information.

We will go all the way to verify such hints fastest

Copyright

Regensburger Straße 7
D-06132 Halle
Tel. +49 345 - 777 55 0
Fax. +49 345 - 777 55 20
E-Mail service@port.de
Internet http://www.port.de

9.1. General Settings . . . 28
9.2. DCF Settings . . . 29
9.3. Network Settings . . . 29
9.4. Color Settings . . . 30
9.5. Font Settings . . . 30

10. NMT Tab . . . 31
10.1. NMT commands . . . 31
10.2. User defined scripts . . . 31
10.3. Network overview . . . 32
10.4. Device Information . . . 32

11. Description Tab . . . 33
11.1. Object Description . . . 33
11.2. Object Description File . . . 33

12. Overview Tab . . . 35
12.1. Index Overview . . . 35

13. PDO Configuration Tab . . . 36
13.1. Configuration of PDOs for the chart . . . 36

14. Process Tab . . . 39

15. PDO Process Tab . . . 40

16. Using Stripcharts . . . 41

17. Extended object configuration . . . 43
17.1. Slider . . . 43
17.2. Slider in a Top Level Window . . . 44
17.3. Switch Box Display of Objects . . . 45

18. Usage of Octet strings . . . 46
18.1. Value entry frame . . . 46
18.2. Octet strings in scripts . . . 47

19. User Specified Tabs . . . 48

19.1. User specified tabs with support of test . . . 48
19.2. User specified tabs without contents . . . 52
19.3. Erasing User Specified Tab-sets . . . 53

20. Data Management with DCF File . . . 54

20.1. Creating a DCF file . . . 54
20.2. Load . . . 55
20.3. Send . . . 55

21. Offline Mode . . . 56

22. Object Data Management with OCF File . . . 57

22.1. Saving of OCF files . . . 57
22.2. Loading . . . 57
22.3. File Format . . . 57

23. Console . . . 58

23.1. Tcl Commands . . . 58
23.2. Scripts . . . 59

24. Scripting with PDOs . . . 60

24.1. Configuration of PDOs in scripts . . . 60
24.2. Transmission of PDOs . . . 61
24.3. Request of PDOs using RTR . . . 61
24.4. Reception of PDOs . . . 61
24.5. Waiting for PDOs . . . 61

25. CAN Message Logging . . . 63

25.1. Scripting-Interface . . . 64

26. SDO Program Download . . . 66

27. Network overview window . . . 68

28. Project Files . . . 69

28.1. Import/Export of CCM project files 69

29. SRDO Tab 70

30. LSS Tab 72

30.1. LSS Mask 72

30.2. LSS Commands 73

31. DSP 402 extension 75

31.1. State machine tab 75

31.2. Profile velocity mode tab 76

31.3. Profile position mode tab 76

31.4. Object extensions 77

32. About & Release Info Dialog 79

32.1. About Dialog 79

32.2. Latest Release Info Dialog 79

33. CDM Command syntax 80

33.1. SDO commands 80

33.2. PDO commands 84

33.3. NMT commands 90

33.4. LSS commands 92

33.5. Other CANopen commands 93

33.6. Test commands 99

33.7. CDM commands 103

33.8. DSP402 commands (DSP402-Extension) 110

34. Appendices 116

34. Appendix 1 — CANopen Commands in Overview 116

34. Appendix 2 — CDM Commands in Overview 118

34. Appendix 3 — Creation of an Object Description 119

34. Literature 120

35. Glossary 121

1. Introduction

1.1. Product Overview

The CANopen Device Monitor (CDM) communicates with CANopen devices in CAN networks by using of various CANopen services. Therewith the Device Monitor supports:

- development,
- diagnostic,
- implementation and configuration

of CANopen devices.

The Device Monitor is a graphical user interface. Various drivers can be integrated in the Device Monitor in dependency of the used CAN interface. The driver interface is called m4d-server or CANopen server. The communication between the graphical user interface and the driver is done by the TCP/IP protocol independent of the location of the Device Monitor and the CANopen server. The user interface and the driver can be at the same computer or on different computers, connected by a LAN, typically an Ethernet TCP/IP connection.

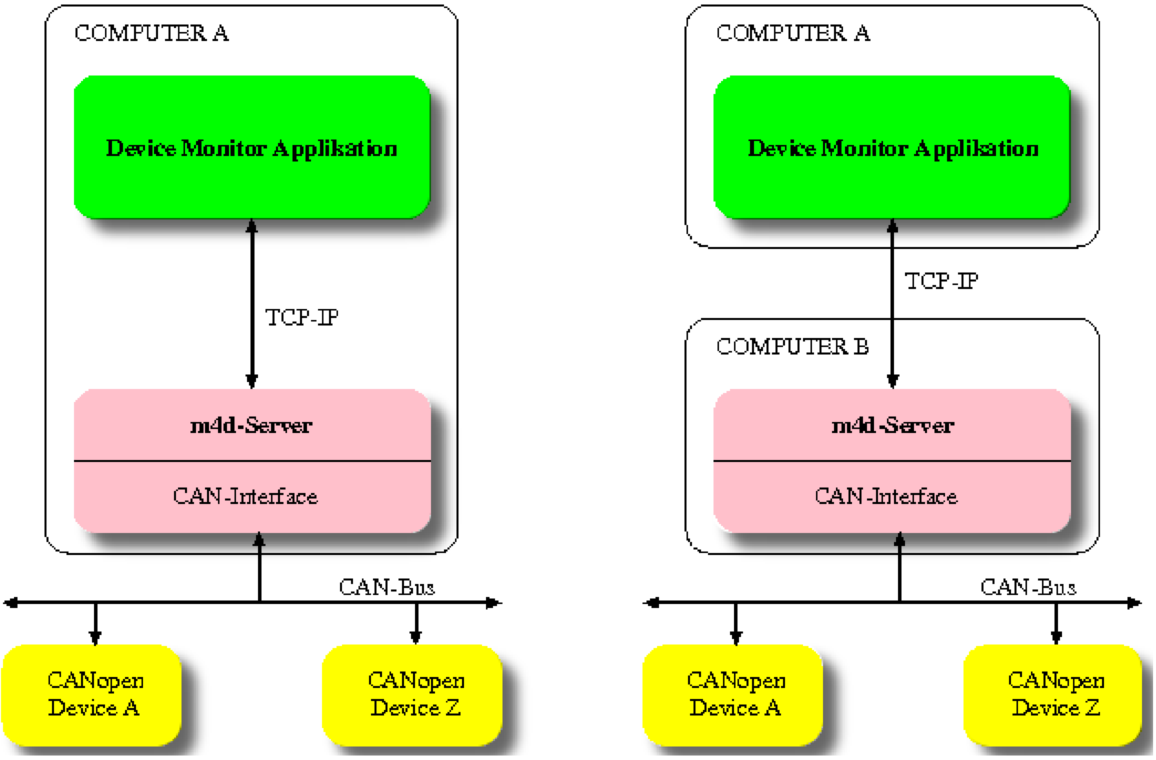


Figure 1: Structure of the CANopen Device Monitor

The CANopen Device Monitor is available in two variations:

- full edition and

- Starter Kit edition.

The StarterKit edition has the following limitations:

- no console for work with scripts,
- unconfigurable SYNC cycle,
- fixed bit rate (125 kBit/s),
- no device configuration by DCF file,
- support of only some fixed node IDs

The CANopen Device Monitor has the capability to execute scripts in full edition. A script can contain variables and control structures just like in every modern structured language. Complete master applications can be developed using the language Tcl/Tk ⟨`http://www.tcl.tk`⟩.

The Device Monitor has the following requirements to the system it is running on:

Operating System:	Windows™, Vista™ UNIX (LINUX)
RAM:	512 MByte
Hard Disk:	25 MByte

The performance of the CANopen Device Monitor depends on the used CAN interface hardware. Especially at high bus load and high baud rates some CAN messages may be lost.

1.2. Conventions

This manual uses the following conventions:

OPERATIONAL	Communication states are written in capital letters.
`tcl_command`	Tcl commands appear in Courier (constant width font). Exceeds the length of a Tcl command line the paper width, this line is finished with a \ (backslash) and continued in the following line.
`example`	Fragments of code and examples appear in Courier (constant width font).
0x<value>	Hexadecimal values are designated by the prefix ’0x’.
<key>	Keys are designated by < >-braces.
<set_value>	Replace set_value by the desired value if this term is a part of a functional description.
[option]	Replace option by an option of the function.

"directory"	Directories are signified with quotation marks.
Console → Save.. → All	the users way through menus

1.3. Support by *port*

port supports the user by telephone hot-line and by seminars. Additionally *port* offers consultations in the whole field of CANopen e.g. network planning, network configuration, selection of devices and CANopen and CANopen Profile implementations.

2. Installation

The installation includes:

- the graphical user interface
- the m4d server and
- a layer 2 driver for the CAN interface.

For the installation the following steps are necessary:

1. Maybe preparing installation steps are necessary depending on the used CAN-Interface. These steps are described in the file **INSTALL** in the m4d directory on the installation CD.

   ! Please read this file before you start the installation.

2. Execute setup.exe.

   • Full Installation: The installation of all software components is happened automatically and menu driven. This includes copying of all manuals.

   • Customized Installation: The selection of software components is possible, which can be installed. For the installation of the CANopen Device Monitor the following components are necessary: CANopen Device Monitor, m4d and layer 2 driver.

3. For the icon on the desktop set the options for the call of the m4d server depending on your application. An overview of the options are given by the help:

   ```
   m4d -h
   ```

   Alternatively the CANopen Device Monitor is able to start the CANopen server (m4d). For additional hints please see section "Hardware Configuration"

4. Define the working directory for the shortcuts on your desktop depending on your application.

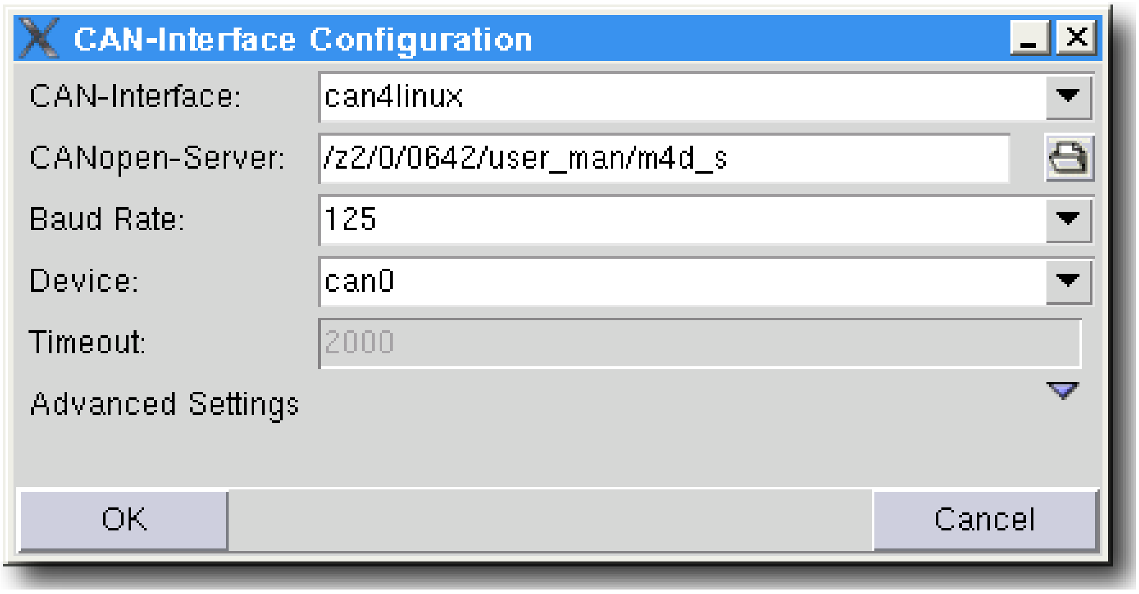
3. Quick Start

3.1. Preparations

- Make sure that the CAN interface, the Layer 2 driver and the CANopen server (m4d) are installed correctly.
- The path to the CANopen server (m4d) must be known (mostly in the CDM directory).
- The preferential method for the start of the CANopen server must be clear.
  - □ possibility 1: the m4d is started by the CDM on your local computer.
  - □ possibility 2: you start the m4d on an arbitrary computer in the network and the CANopen Device Monitor connects to the m4d
- The name of the used CAN interfaces must be known. E.g.:
  - □ can0, can1 using can4linux
  - □ CHAN00, CHAN01 at the use of CPC hardware
  - □ Lpy2Pp and additional unit/device parameter at LevelX hardware

3.2. First program start

At the first start of the CANopen Device Monitor the following dialog window appears, which can vary depending on operating system and available hardware.



CAN-Interface: can4linux

CANopen-Server: /z2/0/0642/user_man/m4d_s

Baud Rate: 125

Device: can0

Timeout: 2000

Figure 2: Hardware configuration dialog

If the CDM shall only connect to a running CANopen server, select at CAN interface "TCP". In this case you need to adjust only the host and the port of the server. The dialogue window can be closed with OK.

In other case the following parameters must be typed in:

- CAN-Interface
- path to the CANopen server
- baud rate of the CAN bus
- name of can interface

If all parameters are configured, the dialog window can be closed with OK. The CANopen Device Monitor starts the CANopen server now and the client-server architecture gets in the background.

All settings in this dialog window can be changed any time via Connection → CAN-Interface.

The CANopen Device Monitor starts the CANopen server now and at success the online status is displayed in the status bar and the background of the entry field turns pink.

! A valid configuration of the hardware interface is saved by the CANopen Device Monitor and can be reused at the next start of the program. Additionally it is possible to activate Extras → Options → Auto-Connect so that the CANopen Device Monitor connects automatically at every start of the program.

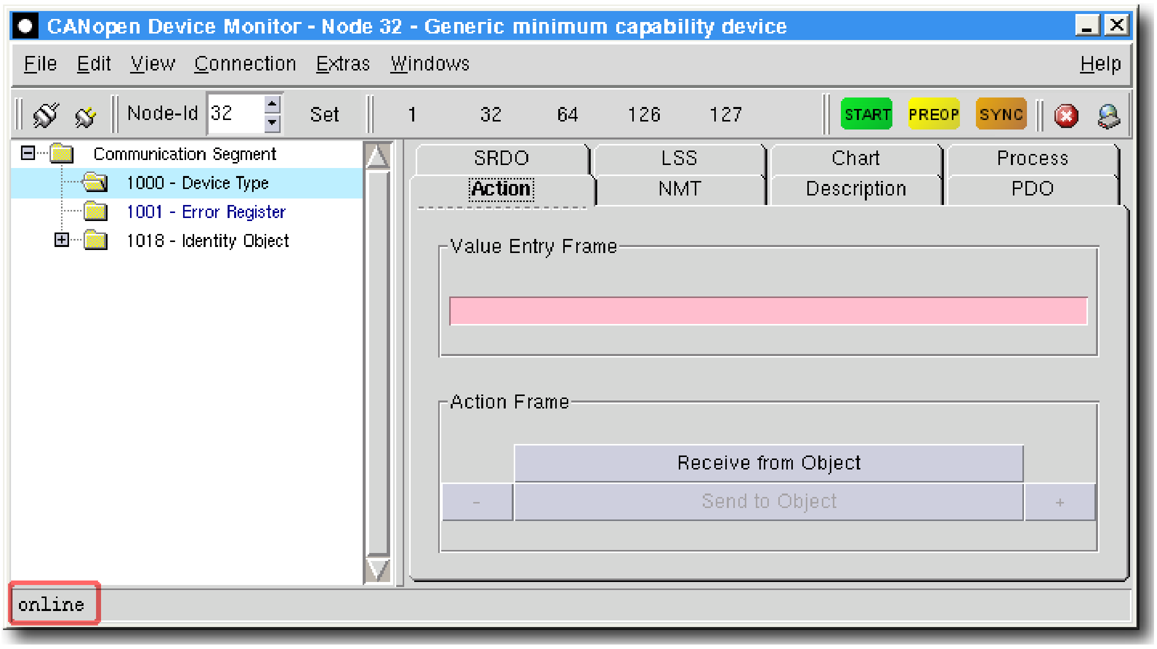


Figure 3: CANopen Device Monitor

3.3. Communication with a CANopen device

Adjust the node ID of the CANopen device in the tool bar first. Please take into account that the choice must be confirmed with "Set".

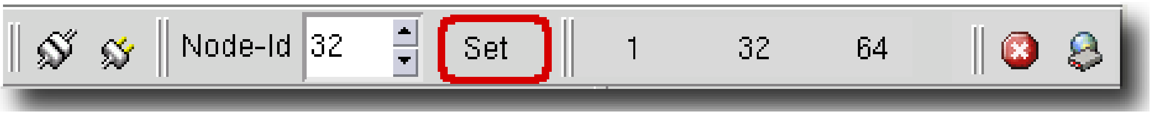
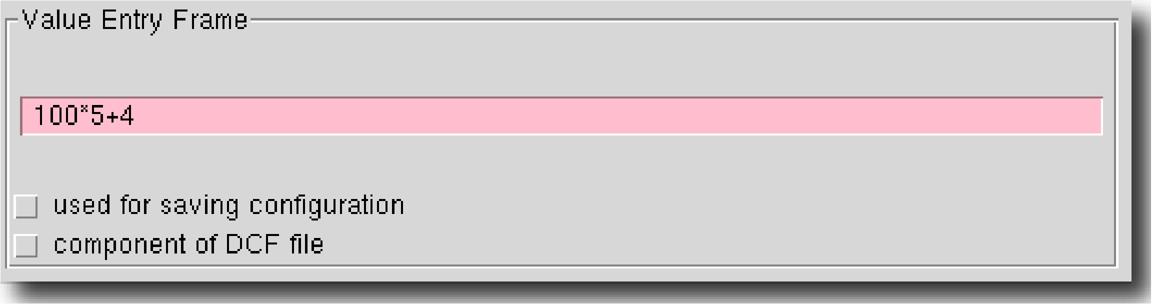


Figure 4: Toolbar

As a second step the EDS file of the device can be loaded via File →Load EDS. Alternatively the mandatory objects can be accessed by the default EDS.

After the selection of an object in the EDS tree you can access it for reading and writing on the "action tab" depending on the access type. When pressing "Send to object" only the expression after the last blank is transmitted. It is possible to write mathematical expressions into the entry field as well. The CANopen Device Monitor calculates them and transmits the result to the device. However, the expression may not contain any blanks.


Value Entry Frame

100*5+4

□ used for saving configuration
□ component of DCF file

Figure 5: Formulas in the entry

The "NMT tab" serves for sending NMT commands. This can be torn off like every tab from the anchorage in the CANopen Device Monitor out into a separate window.

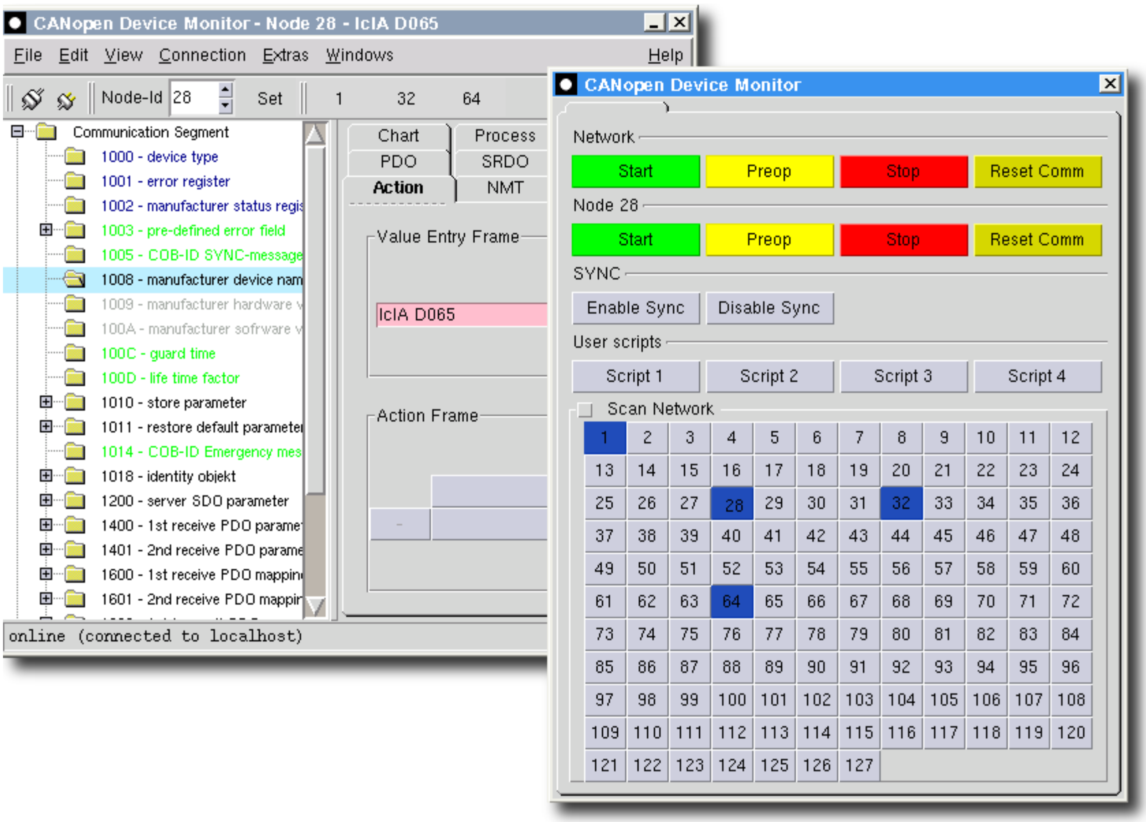


Figure 6: Separation of a tab

The upper command bar serves for sending the NMT commands to the whole network and with the lower one the current node is addressed. The current node can be changed quickly by the node buttons in the lower part of the NMT tab.

4. Configuration

The Device Monitor can be adapted to the personal environment. This adaptation can be stored in the file "<working directory>\cdm.rc" with the exception of the CAN bit rate. A pattern of the file cdm.rc is located in the "<CDM directory>". The bit rate is specified as start-up option of the m4d-server. It is valid for the CANopen Network, not only for one Device Monitor instance. During start-up the Device Monitor loads the file "<working directory>\cdm.rc".

Please use an editor that is capable of handling LF line endings correct. Do not use notepad.exe.

! The following sub section can be skipped, if the CANopen server (m4d) is started by the CANopen Device Monitor. For additional hints please see section "Hardware Configuration"

4.1. Setting of the CAN Bitrate

The CAN bit rate will be set at the call of the m4d-server:

```
Windows™: A:\ProgramFiles\CANopen> m4d_<driver>_s.exe -b <CAN bit rate> -S
Unix: /usr/bin $ m4d_s -b <CAN bitrate>  -S
```

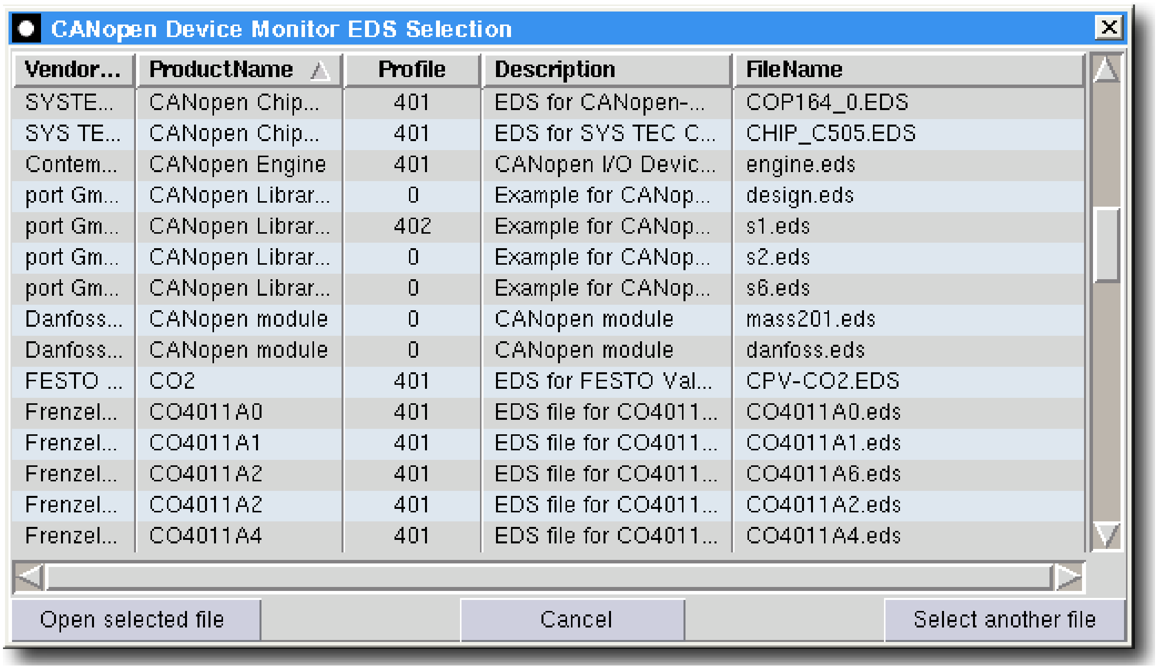
All CAN bit rates defined in CANopen are available.

4.2. Loading of the Object Dictionary

Every device manages its parameters in an object dictionary. The object dictionary can be stored in a CANopen defined format, the Electronic Data Sheet or EDS file. Every device has its own **EDS file**. During start-up the Device Monitor can load a default EDS file.

If you exchanged CANopen Devices you can load the EDS File of the CANopen device: File → Load EDS

The management of the EDS files is simplified by the EDS repository. The repository is a directory that contains all EDS files. Via Extras → Options → EDS-Repository the EDS repository can be configured. If the EDS repository is configured correctly, the following dialog appears when loading an EDS file via File → Load EDS



Vendor...	ProductName	Profile	Description	FileName
SYSTE...	CANopen Chip...	401	EDS for CANopen-...	COP164_0.EDS
SYS TE...	CANopen Chip...	401	EDS for SYS TEC C...	CHIP_C505.EDS
Contem...	CANopen Engine	401	CANopen I/O Devic...	engine.eds
port Gm...	CANopen Librar...	0	Example for CANop...	design.eds
port Gm...	CANopen Librar...	402	Example for CANop...	s1.eds
port Gm...	CANopen Librar...	0	Example for CANop...	s2.eds
port Gm...	CANopen Librar...	0	Example for CANop...	s6.eds
Danfoss...	CANopen module	0	CANopen module	mass201.eds
Danfoss...	CANopen module	0	CANopen module	danfoss.eds
FESTO ...	CO2	401	EDS for FESTO Val...	CPV-CO2.EDS
Frenzel...	CO4011A0	401	EDS file for CO4011...	CO4011A0.eds
Frenzel...	CO4011A1	401	EDS file for CO4011...	CO4011A1.eds
Frenzel...	CO4011A2	401	EDS file for CO4011...	CO4011A6.eds
Frenzel...	CO4011A2	401	EDS file for CO4011...	CO4011A2.eds
Frenzel...	CO4011A4	401	EDS file for CO4011...	CO4011A4.eds

Figure 13: EDS choice box

4.3. Device Specific Settings

If the Device Monitor loads a new EDS file, at start-up or if selected by File → Load EDS it looks also if a device description or device configuration file with the same name as the EDS file but the extension .rc is available. This file has to be in the same directory like the EDS file.

EXAMPLE:
device_v1_1.eds and device_v1_1.rc → right
device1_1.eds and device_v1_1.rc → wrong

This configuration file is used for the storage of device specific settings. That can be local definitions or assignments, like the definition of sliders for special objects but also statements with CANopen commands to initialize objects within the CANopen devices object dictionary.

After the installation of the Device Monitor the CDM directory contains some examples for device configuration files.

4.4. Start-Up Scripts

A Tcl/Tk-script which is loaded at the start of the CANopen Device Monitor can be specified with the global variable `autoExecScript` in the configuration file cdm.rc.

EXAMPLE:

```
# specify start-up script
set autoExecScript "demo.tcl"
```

This script (e.g. `demo.tcl`) is loaded after the complete start-up of the CANopen Device Monitor. That means that start-up actions like loading the last project file or automatically connect to the network are carried out before loading the script.

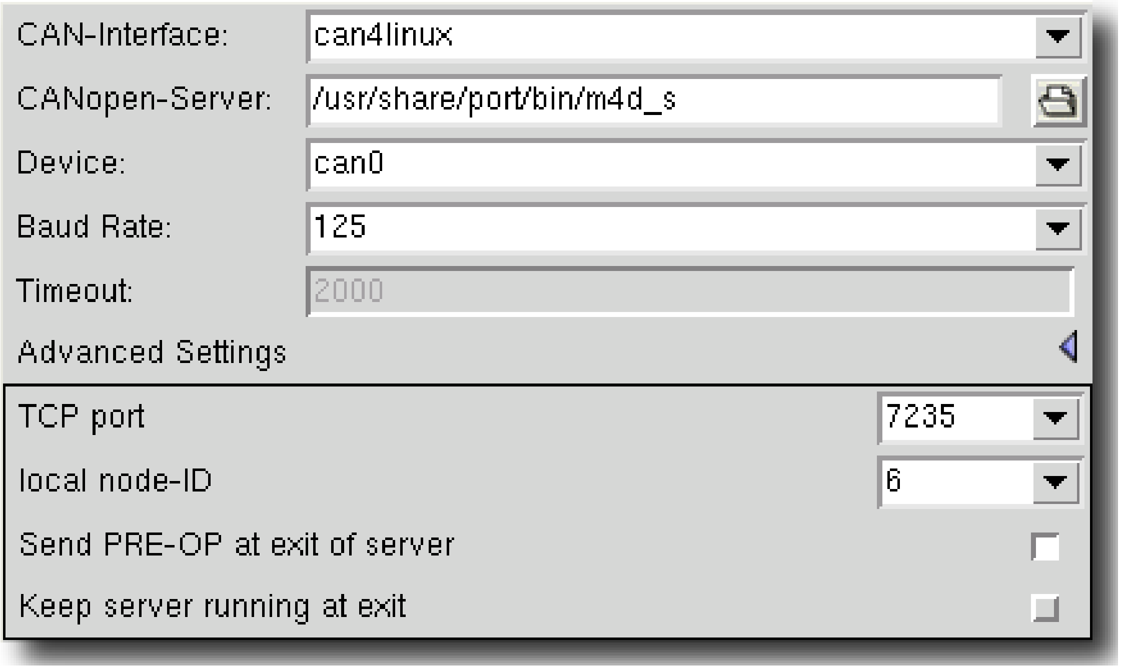
5. CAN-Interface Configuration

As extension to the previous described method the CANopen Device Monitor is also capable to start the CANopen-Server by itself. Thereby the CANopen Device Monitor handles the start, the connection establishment, and the closing of the CANopen-Server (m4d) automatically.

At the first start of the CANopen Device Monitor after the installation the configuration dialog opens automatically. After that the hardware configuration can be opened by Connection → Configure CAN-Interface

The other method with a separate start of the CANopen-Server is still possible, so that its advantages like a remote-control of device via a TCP/IP network can be used, too.

5.1. Configuration Dialog



CAN-Interface: can4linux

CANopen-Server: /usr/share/port/bin/m4d_s

Device: can0

Baud Rate: 125

Timeout: 2000

Advanced Settings

TCP port 7235

local node-ID 6

Send PRE-OP at exit of server □

Keep server running at exit □

Figure 14: Hardware configuration dialog

Is TCP selected as "CAN-Interface", so the CANopen-Server has to be started separately and all other options in the dialog are disabled. The desired options have to be passed to the CANopen-Server at its start.

The following options are available for all supported hardware interfaces:

Option	Description
CANopen-Server	path to the CANopen-Server (m4d)
Baud Rate	CAN Baudrate [kbit/s]
TCP port	TCP-port used for the communication between the CDM and the CANopen-Server (m4d)

local node-ID	Node-ID of the CANopen-Server (m4d)
Send PREOP at exit	automatic transmission of the NMT command ENTER PREOPERATIONAL to all devices when the CANopen-Server is shut down
Keep server running at exit	the CANopen-Server will not be closed when the CDM is closed

Depending on the hardware interface some additional options like device, channel, board or unit can be present. These options distinguish the connected device or the desired communication channel.

The CANopen Device Monitor searches for all installed drivers and CANopen-Server and offers only the installed drivers at the option CAN-Interface. Mostly it's only one driver plus TCP as default option.

6. Object Dictionary Accesses

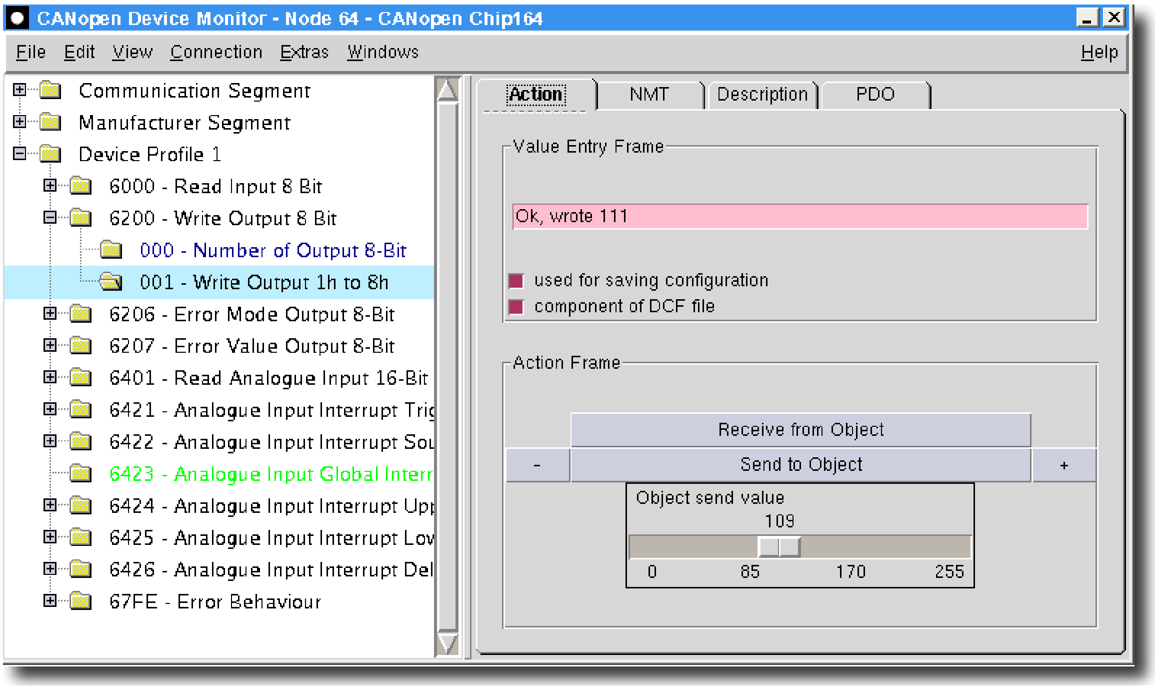
The object dictionary is the data interface between the CANopen Device Monitor and the CANopen device. The CANopen Device Monitor can access every object in the object dictionary for reading or writing by index/subindex. CANopen subdivides the objects into various groups:

Index	Objects
0000h - 0FFFh	common types
1000h - 1FFFh	communication objects
2000h - 5FFFh	manufacturer specific objects
6000h - 9FFFh	profile objects
A000h - AFFFh	objects for network variables

The CANopen Device Monitor shows the object dictionary loaded from the Electronic Data Sheet (EDS file) of the device in a tree-structure.

An EDS can be loaded for each device in the network. After the selection of a device its object dictionary is displayed and communcation with the device is possible.

Using the tab Action objects can be accessed by SDO transfers.


CANopen Device Monitor - Node 64 - CANopen Chip164
File
Edit
View
Connection
Extras
Windows
Help
Communication Segment
Manufacturer Segment
Device Profile 1
6000 - Read Input 8 Bit
6200 - Write Output 8 Bit
000 - Number of Output 8-Bit
001 - Write Output 1h to 8h
6206 - Error Mode Output 8-Bit
6207 - Error Value Output 8-Bit
6401 - Read Analogue Input 16-Bit
6421 - Analogue Input Interrupt Trig
6422 - Analogue Input Interrupt Sou
6423 - Analogue Input Global Interr
6424 - Analogue Input Interrupt Upp
6425 - Analogue Input Interrupt Low
6426 - Analogue Input Interrupt Del
67FE - Error Behaviour
Action
NMT
Description
PDO
Value Entry Frame
Action Frame
Receive from Object
-
Send to Object
+
0
85
170
255

Ok, wrote 111

■ used for saving configuration
■ component of DCF file

Object send value
109

Figure 15: Tab Action

Write object

select the object in the tree

set value in Action tab→ Value Entry Frame execute SDO transfer by Action → Send to Object

Read object

select the object in the tree

execute SDO transfer by Action → Receive from Object

the received value is shown in Action → Value Entry Frame

Because of the fact that reading an object may trigger an unwanted reaction an object is only read on request. But if Extras → Options → Read object on selection is active the selection of an object triggers the SDO transfer to read this object.

Value Entry Frame

Hexadecimal values are designated by the prefix 0x (example: 0x10).

Enter strings consisting of one or more parts without quotation marks. Spaces at the end of the string will be ignored.

Is this field empty, the value 0 will be sent.

<DEL> deletes the contents of this field.

<ENTER> sends the value.

<Shift ENTER> If an URL is in this entry, the web browser is started.

The last word of the value in the entry field is evaluated as a mathematical expression before sending it with an SDO transfer.

Try to put something like: *0x180+10* in it.

Component of DCF file

By this checkbutton the object is marked for DCF file handling (see chapter "Data Management with DCF File").

Used for saving configuration

If this check box is activated, the CANopen Device Monitor marks this object for storing its value in a object configuration (*.ocf) file (see chapter "Object Data Management with ocf File").

Cyclic update

If this check box is activated, the value is read and updated cyclically. Active bit boxes and slider displaying the value of this object are updated too.

Receive from Object

The object value will be requested by SDO.

Send to Object

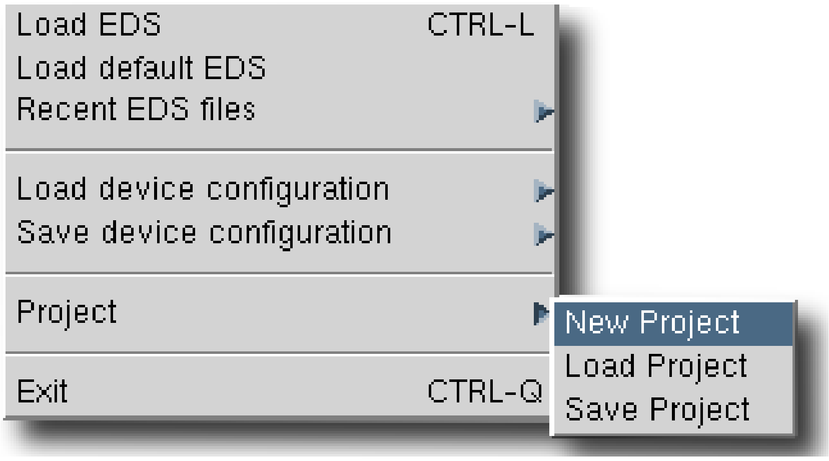
The object value will be set (written) by SDO.

- The value in the field Value Entry Frame will be decremented by 1. Thereafter it will be sent to the device automatically.

+ The value in the field Value Entry Frame will be incremented by 1. Thereafter it will be sent to the device automatically.

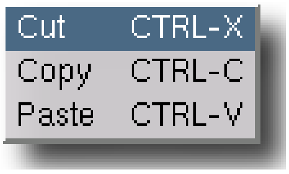
7. Menu Structure

7.1. File



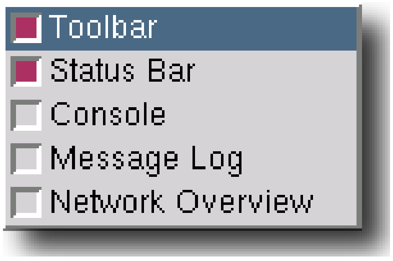
Load EDS	Loads an EDS file
Load default EDS	Loads an EDS file with mandatory objects
Recent EDS files	List of recently used EDS files
Load device configuration	Loads OCF or DCF files
Save device configuration	Saves OCF or DCF files
New Project	Starts a new project
Load Project	Loads an existing project
Save Project	Saves a project
Exit	Exits CANopen Device Monitor

7.2. Edit



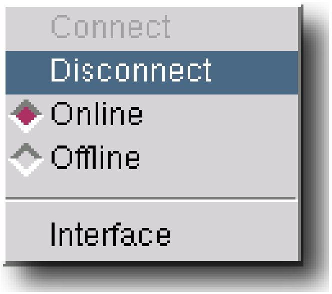
Cut	Cuts selected text into clipboard
Copy	Copies selected text into clipboard
Paste	Pastes text from clipboard

7.3. View



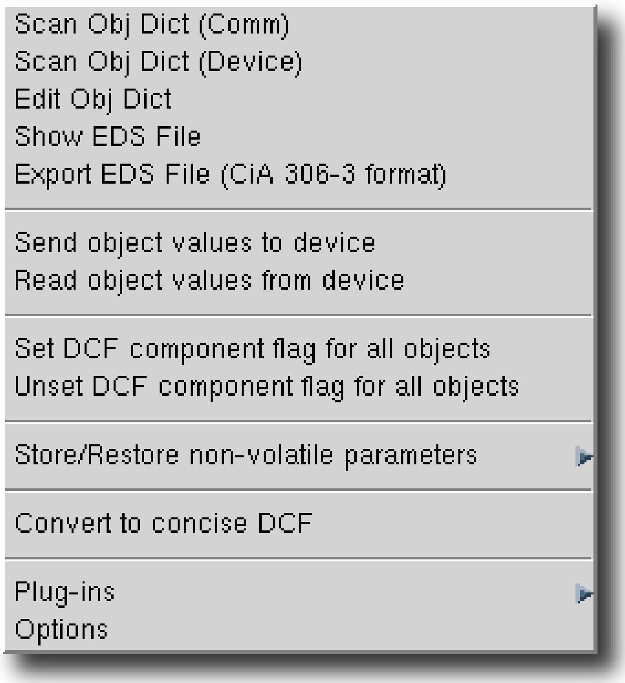
Toolbar	Toggles the view of the toolbar
Status Bar	Toggles the view of the statusbar
Console	Toggles the view of the console (only full version)
Message Log	Toggles the view of the log window
Network View	Toggles the view of the network overview

7.4. Connection



Connect	Connects to the CANopen server resp. starts the CANopen server
Disconnect	Closes the connection to the CANopen server resp. closes the CANopen server
Online	Read and write accesses are performed directly with the device
Offline	Read and write accesses are performed with the DCF data buffer
Interface	Opens the dialog to configure the CAN interface

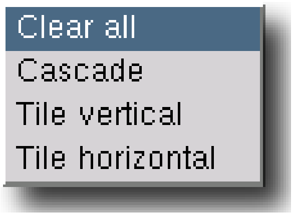
7.5. Extras


Scan Obj Dict (Comm)
Scan Obj Dict (Device)
Edit Obj Dict
Show EDS File
Export EDS File (CiA 306-3 format)
Send object values to device
Read object values from device
Set DCF component flag for all objects
Unset DCF component flag for all objects
Store/Restore non-volatile parameters
Convert to concise DCF
Plug-ins
Options

Scan Obj Dict (Comm)	Scans the communication profile segment of the object dictionary
Scan Obj Dict (Device)	Scans the device profile segment of the object dictionary
Edit Obj Dict	Opens a simple OD-Editor. It is e.g. useful to add dynamic OD entries.
Export EDS File	Exports a scanned object dictionary as EDS file.[1]
Send object values to device	Transmits the values of all objects with DCF-component-flag to the device.
Read object values from device	Reads the values of all objects with DCF-component-flag from the device.
Store/Restore non-volatile parameters	Store or restores the configuration of the device in its non-volatile memory.
Convert to concise DCF	Converts DCF-files into the concise format
Plug-ins	Menu to load plug-ins
Options	Opens the option dialog

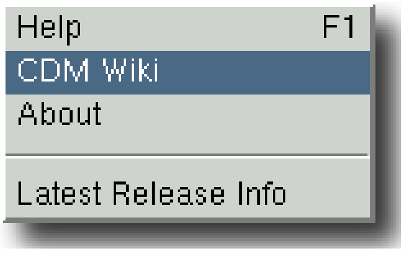
[1] The exported file can be read again by the CANopen Device Monitor, but it is no complete EDS file according to the standard.

7.6. Windows



Clear All	Clears the console and the log window	except of the main window
Cascade	Cascades all open windows	
Tile vertical	Tiles all open windows vertically	
Tile horizontal	Tiles all open windows horizontally	

7.7. Help



Help	Shows the on-line help
CDM Wiki	Opens the CDM wiki in a web browser
About	Shows version and licence information
Latest Release Info	Fetch information about updates from the internet

8. Toolbar

The toolbar is explained in the following illustration.

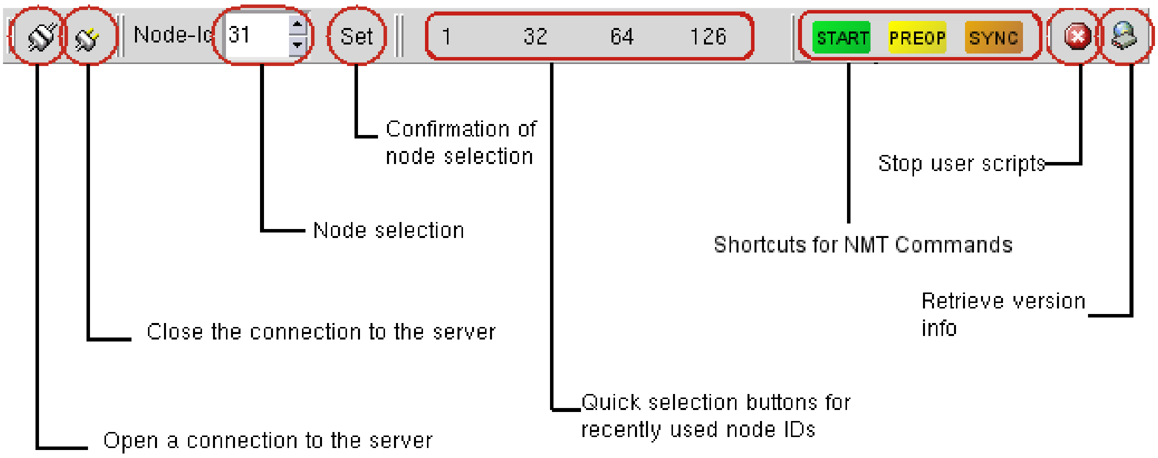
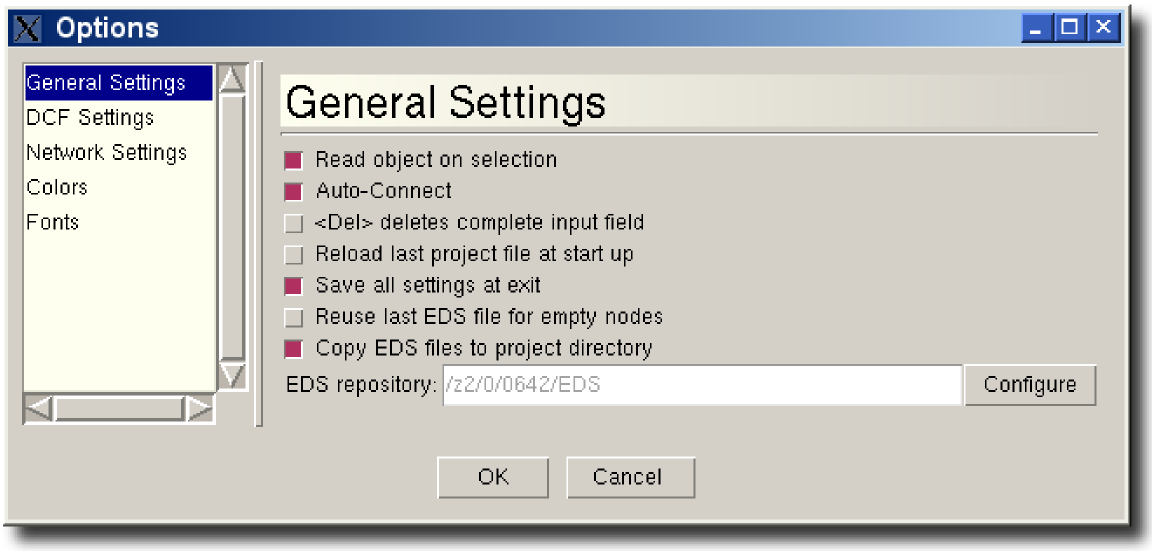

Node-Id
31
Set
1
32
64
126
START
PREOP
SYNC
Confirmation of node selection
Stop user scripts
Node selection
Shortcuts for NMT Commands
Retrieve version info
Close the connection to the server
Quick selection buttons for recently used node IDs
Open a connection to the server

Figure 16: Toolbar

9. Options

9.1. General Settings



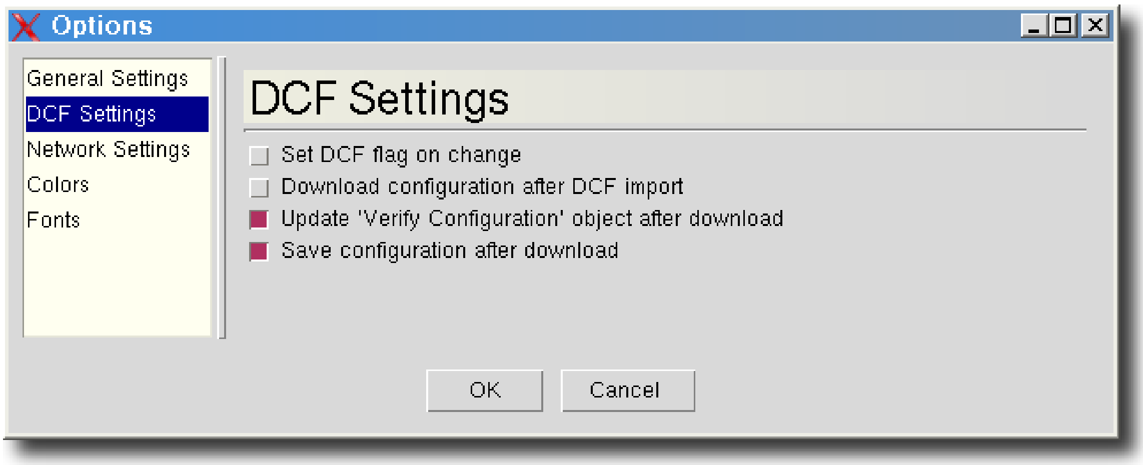
General Settings

- ■ Read object on selection
- ■ Auto-Connect
- □ <Del> deletes complete input field
- □ Reload last project file at start up
- ■ Save all settings at exit
- □ Reuse last EDS file for empty nodes
- ■ Copy EDS files to project directory

EDS repository: /z2/0/0642/EDS

Read object on selection	*read object immediately on selection*
Auto-Connect	*Connect to server at start-up*
<Del> deletes complete entry field	*<Del> key deletes complete input field in value entry frame.*
Reload last project file at start up.	*The lastly used project file is opened automatically at start up of the CANopen Device Monitor.*
Save all settings at exit	*All settings (options and connection settings) are saved automatically.*
Reuse last EDS for empty nodes	*The last EDS file is used for other node-IDs without assigned EDS file.*
Copy EDS files to project directory	*The EDS files are copied from the repository to the project folder, if it is saved.*
EDS repository	*Configuration of the directory for the EDS repository*

9.2. DCF Settings



Options

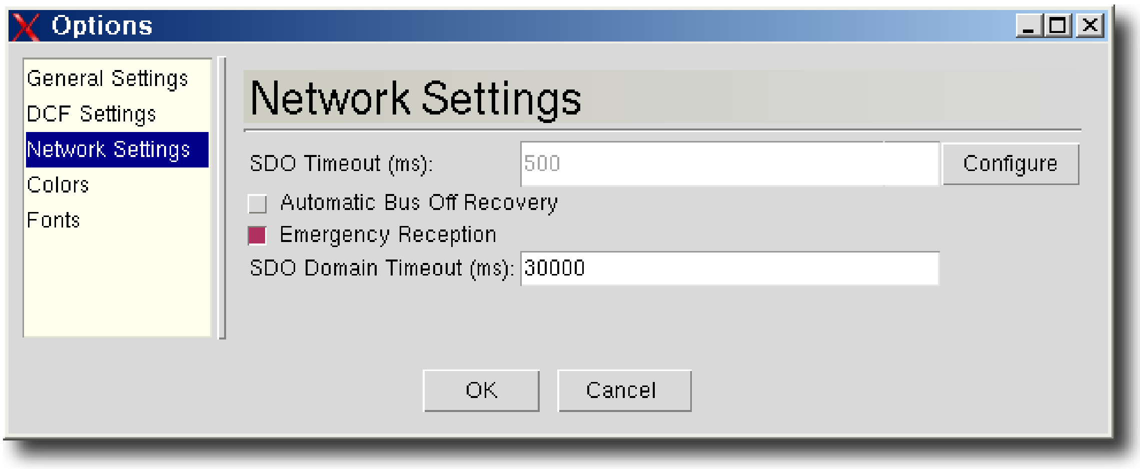
DCF Settings

- ☐ Set DCF flag on change
- ☐ Download configuration after DCF import
- ☑ Update 'Verify Configuration' object after download
- ☑ Save configuration after download

Set DCF flag on change	*sets the DCF component flag at change of an object*
Download configuration after DCF import	*Automatic download of the configuration after DCF import*
Save configuration after download	*Automatic saving of the configuration after download*

Update 'Verify Configuration' object after download	*Automatic update of the object 0x1020 with the current configuration time*

9.3. Network Settings



Options

Network Settings

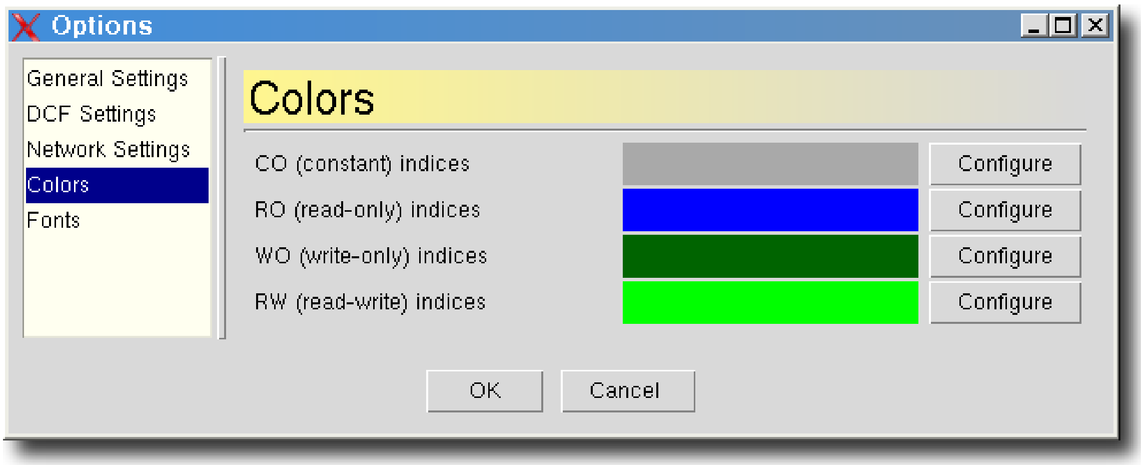
SDO Timeout (ms): 500

- ☐ Automatic Bus Off Recovery
- ☑ Emergency Reception

SDO Domain Timeout (ms): 30000

SDO Timeout (ms)	*SDO timeout in ms*
Automatic Bus Off Recovery	*Automatic Bus-On after a Bus-Off event*
Emergency Reception	*Reception of Emergency messages by the CANopen Device Monitor and forwarding to Emergency handler functions*
SDO Domain Timeout (ms)	*SDO timeout for domain transfers in ms*

9.4. Color Settings



Colors

CO (constant) indices

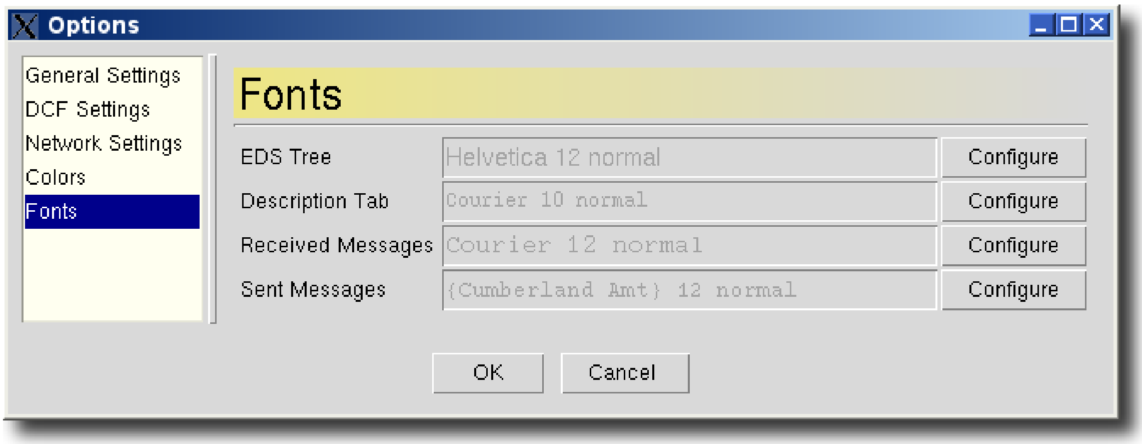
RO (read-only) indices

WO (write-only) indices

RW (read-write) indices

This dialogue allows the configuration of specific colors for objects with different access types.

9.5. Font Settings



Fonts

EDS Tree: Helvetica 12 normal

Description Tab: Courier 10 normal

Received Messages: Courier 12 normal

Sent Messages: {Cumberland Amt} 12 normal

This dialogue allows the configuration of specific fonts for different GUI elements.

10. NMT Tab

10.1. NMT commands

For the execution of NMT services this tab provides some buttons:

Button	Description
Start Network	Set all nodes in the state OPERATIONAL.
Start Node	Set the node with the active node-id in the state OPERATIONAL.
Preop Network	Set all nodes in the state PRE-OPERATIONAL.
Preop Node	Set the node with the active node-id in the state PRE-OPERATIONAL.
Stop Network	Set all nodes in the state STOPPED.
Stop Node	Set the node with the active node-id in the state STOPPED.
Reset Comm	Send the NMT command Reset Communication.
Reset Node	Send the NMT command Reset Node.
Enable Sync	Starts the cyclic transmission of the SYNC message
Disable Sync	Stops the cyclic transmission of the SYNC message
One Sync	Sends one SYNC message

The control of the state machine is also possible by the console.

10.2. User defined scripts

By pressing the buttons "Script "1 to "Script 4" scripts with the file names script1.tcl - script4.tcl are started, if this files can be found in the current working directory or in the program directory. "Test" starts the script *t_start.tcl*, if it exists. Modify these scripts to match your needs and use them to automate repetitious tasks, like configuring a device. Tooltips over each button show the first line of the corresponding script file. So the first line of a script contains a comment about the content of the scripts. Additionally the last word of the first line of a script can be a valid color definition (e.g. red or #ffaa11) to modify the background color of the specific button. Within these scripts all CDM-commands are available.

This function is only available in the full version. Using the eval version only integrated demo scripts can be loaded.

10.3. Network overview

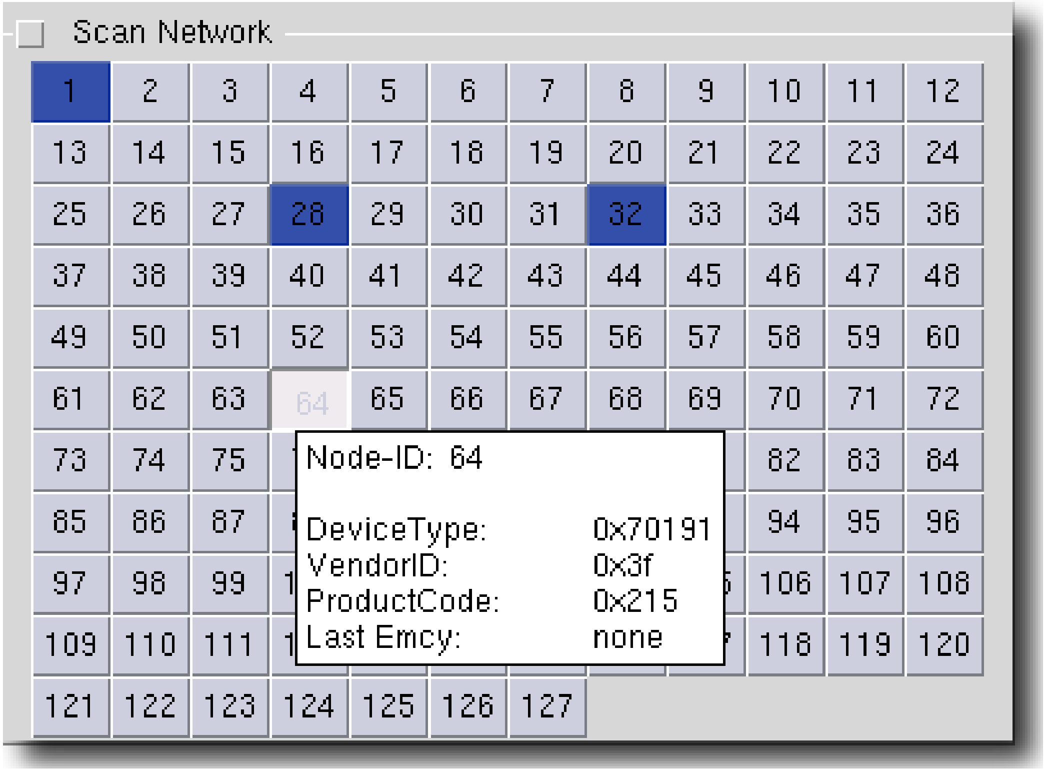
After scanning the network the buttons for the nodes get different colors.

Color	Meaning
background color	no node found
blue	node found
yellow	active node during network scan

To change the active node simply press the button with the desired node-id. If an EDS-file has been loaded already for this node, it is displayed in the EDS tree. Otherwise an EDS-file for this device can be loaded.

For all nodes different EDS files can be loaded.

10.4. Device Information



Node-ID: 64

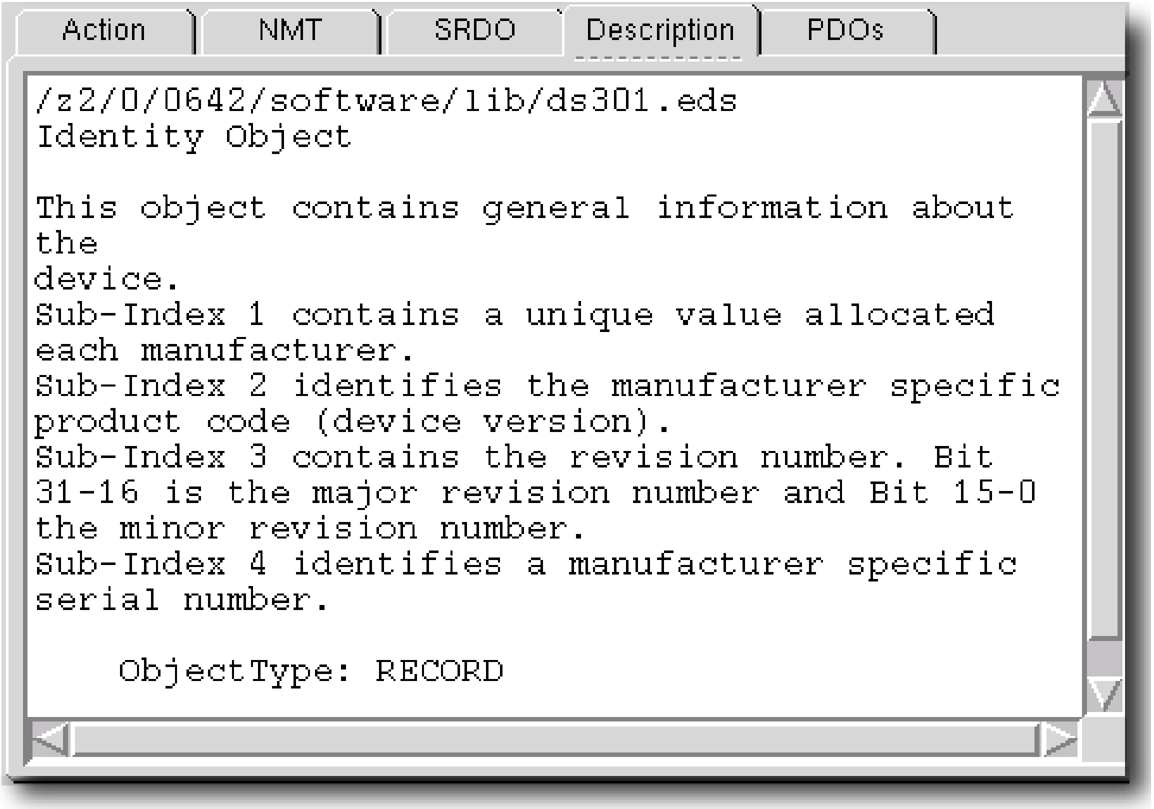
DeviceType: 0x70191
VendorID: 0x3f
ProductCode: 0x215
Last Emcy: none

Figure 17: Device information

When moving the mouse over found nodes device information are displayed as a tool tip.

11. Description Tab

11.1. Object Description



```
/z2/0/0642/software/lib/ds301.eds
Identity Object

This object contains general information about
the
device.
Sub-Index 1 contains a unique value allocated
each manufacturer.
Sub-Index 2 identifies the manufacturer specific
product code (device version).
Sub-Index 3 contains the revision number. Bit
31-16 is the major revision number and Bit 15-0
the minor revision number.
Sub-Index 4 identifies a manufacturer specific
serial number.

    ObjectType: RECORD
```

Figure 18: object description

The object description contains additional information about the selected index of the object dictionary. Beside the actual description the object code for complex objects or the data type and the default value is shown, too. The object description for each index is read from the object description file.

11.2. Object Description File

Because of the restrictions of the old EDS format (according to DSP-306), it is not possible to add object descriptions to an EDS file. Therefore this additional information is stored in a separated object description file. See file structure below:

```
index1:
object name 1

description line 1
description line 2
description line n

index2:
```

```
object name 2

description line 1
description line 2
description line n
```

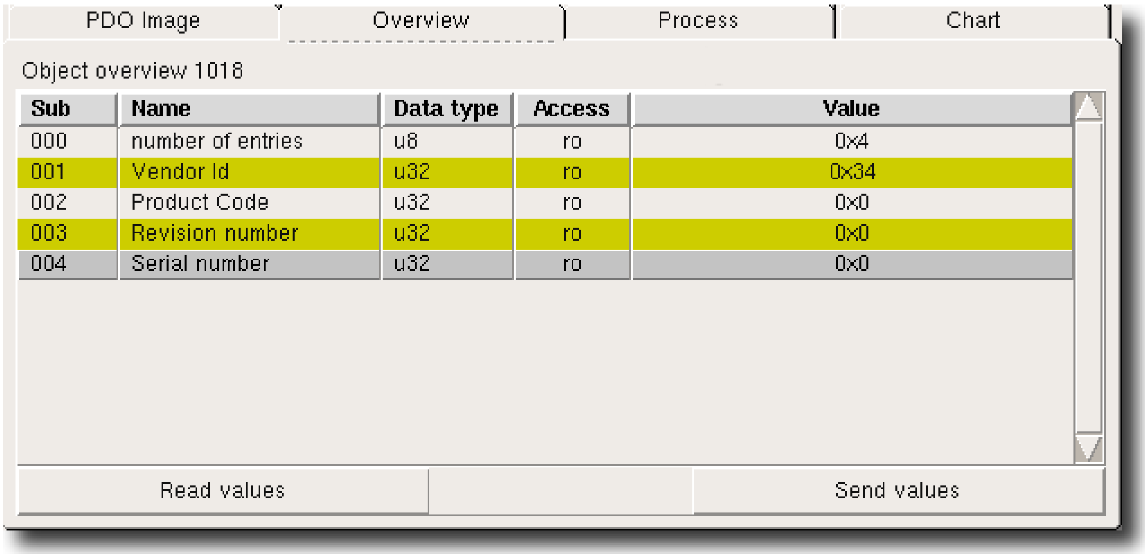
The data format for the indices is hexadecimal without leading "0x". An example object description file is `lib/ds301.txt` with descriptions for the objects from the communication profile. When an EDS file is loaded, the CANopen Device Monitor looks for a file with the same name as the EDS but with the extension `.txt` If such a file is found, it is loaded as object description file for this EDS. Otherwise only the data type or the object code is displayed.

The CANopen DesignTool by *port* creates object description files automatically.

12. Overview Tab

12.1. Index Overview

With the object overview tab all sub indices of an array or record an be read or written at once.



Object overview 1018

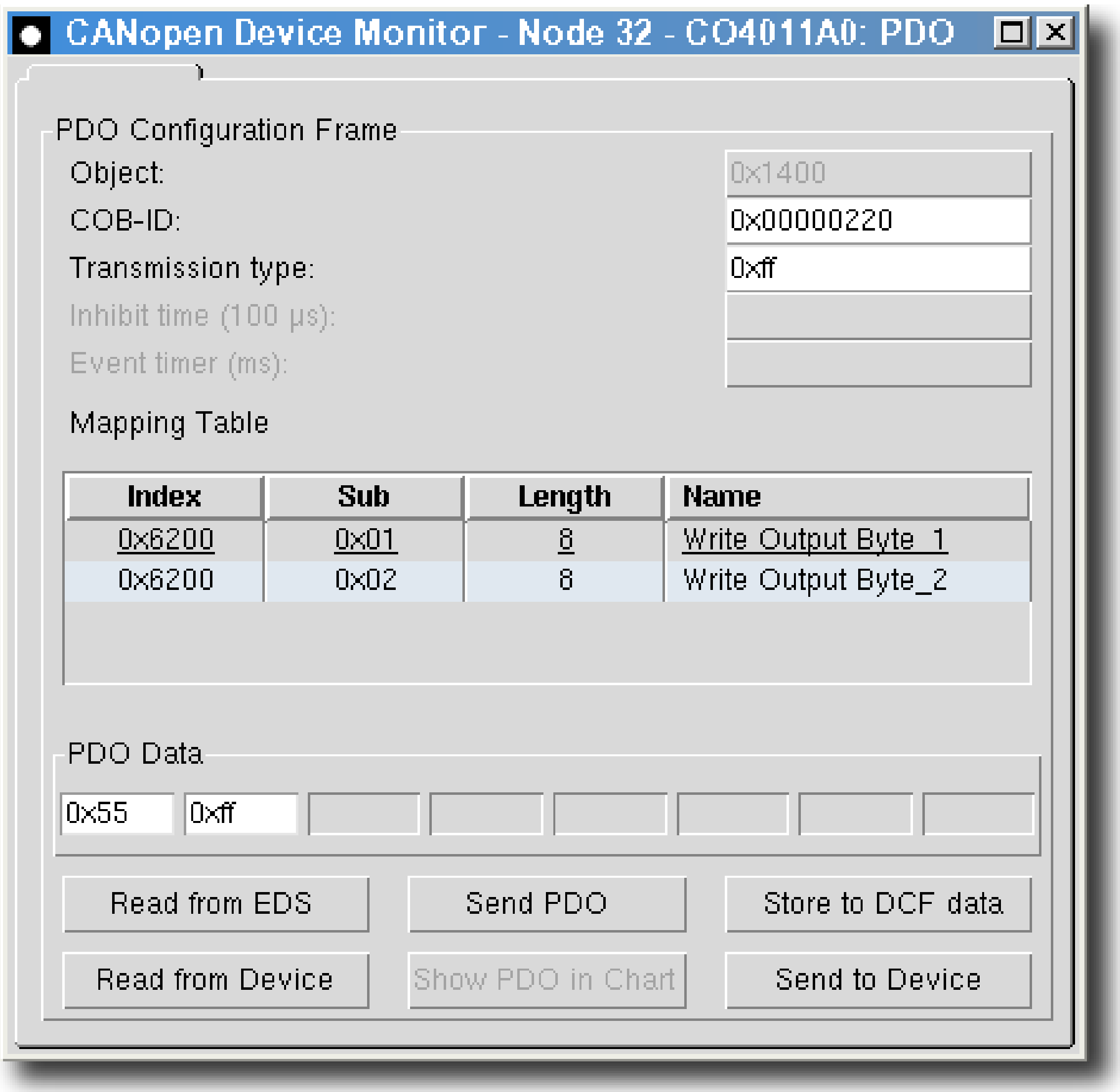
Sub	Name	Data type	Access	Value
000	number of entries	u8	ro	0x4
001	Vendor Id	u32	ro	0x34
002	Product Code	u32	ro	0x0
003	Revision number	u32	ro	0x0
004	Serial number	u32	ro	0x0

Figure 19: Index overview

Some rescritions exist when reading or writing sub indices. Sub indices with the data type "domain" cannot be read or written and PDO, PDO mapping and SRDO objects cannot be written from the overview tab. For these objects it is required to meet a specific order when writing to them, but they can be configured by the PDO configuration tab or the SRDO configuration tab (requires Safety PlugIn).

13. PDO Configuration Tab

The PDO tab simplifies the configuration of PDOs. At the selection of a PDO object in the EDS tree, the mask is updated with values from the EDS. The PDO parameters simply can be adjusted over the mask. New objects can be moved from the EDS tree into the mapping table by drag&drop. A double click deletes them within the mapping table. The table is unalterable at a static mapping.



CANopen Device Monitor - Node 32 - CO4011A0: PDO

PDO Configuration Frame

Object: 0x1400

COB-ID: 0x00000220

Transmission type: 0xff

Inhibit time (100 µs):

Event timer (ms):

Mapping Table

Index	Sub	Length	Name
0x6200	0x01	8	Write Output Byte_1
0x6200	0x02	8	Write Output Byte_2

PDO Data

0x55 0xff

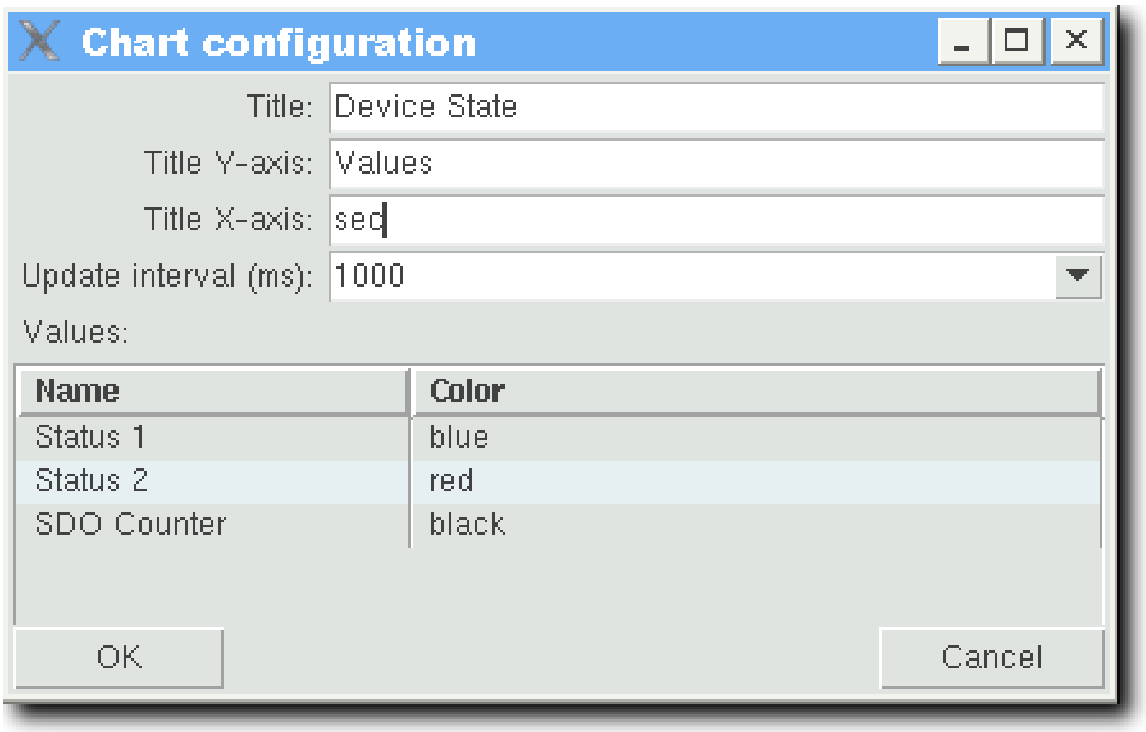
Figure 20: PDO Tab

The following table describes the buttons in the lower part of the mask.

Option	Description
Read from EDS	Reads the values from the EDS file
Read from Device	Reads the current values from the device
Send to Device	Transfers the entered values to the device
Store to DCF data	Store the entered values into the DCF data buffer
Send PDO	Sends the current PDO to the device. The values of the PDO are taken from the entries above. Take into account that PDOs only can be sent or received in the state OPERATIONAL. Only at RPDOs of the device.
Show PDO in Chart	A PDO Indication function is configured so that the values of this PDO are displayed in the chart Tab. The device must be configured correspondingly before. Take into account that PDOs only can be sent in the state OPERATIONAL by the device. Only at TPDOs of the device.

13.1. Configuration of PDOs for the chart

After pressing the button "Show in Chart" the following dialog window is opened.



Title: Device State

Title Y-axis: Values

Title X-axis: sec

Update interval (ms): 1000

Values:

Name	Color
Status 1	blue
Status 2	red
SDO Counter	black

Figure 21: Chart configuration mask

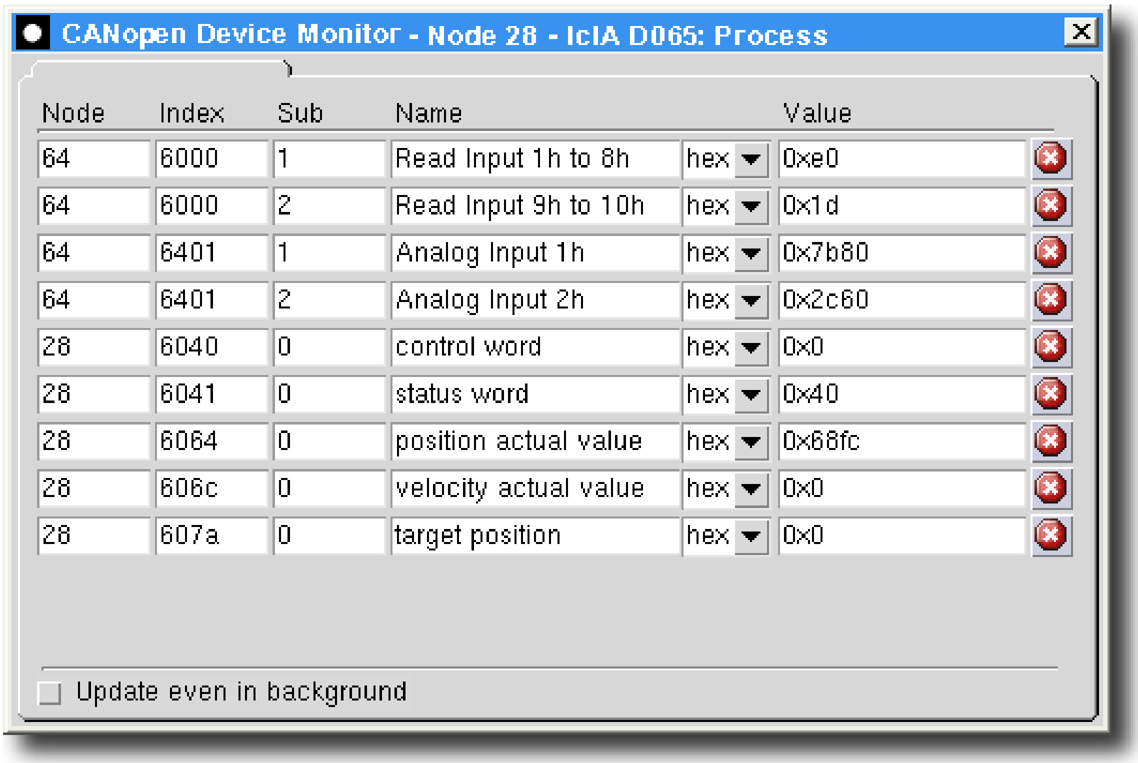
The title of the chart and of the axes and the names of the data can be configured there.

If the **update interval** is not 0, the chart is updated by a time-trigger mechanism. I.e. arriving data are written into a buffer and after a certain amount of time the values of the buffer are displayed in the chart. If the update interval is 0, the chart is updated at every arriving PDO. It it's an asynchronous PDO, the information about the time of the arrival of the data is lost.

At **Color** the color of a signal can be configured. Every color that is supported by Tcl/Tk can be used. Otherwise the colors can be specified in hexadecimal notation like #ffaa11.

14. Process Tab

The process image Tab serves the observation of process quantities of different nodes. The values of the individual objects are queried and updated cyclically by SDO every 1000 ms (per default). Normally, the values are read only if the tab is active. To update the values in every cycle, activate *Update even in background*.



Node	Index	Sub	Name		Value
64	6000	1	Read Input 1h to 8h	hex	0xe0
64	6000	2	Read Input 9h to 10h	hex	0x1d
64	6401	1	Analog Input 1h	hex	0x7b80
64	6401	2	Analog Input 2h	hex	0x2c60
28	6040	0	control word	hex	0x0
28	6041	0	status word	hex	0x40
28	6064	0	position actual value	hex	0x68fc
28	606c	0	velocity actual value	hex	0x0
28	607a	0	target position	hex	0x0

□ Update even in background

Figure 23: Process image

Add single objects by Drag & Drop from the EDS tree. The small button besides the value of the object serves for deleting. Whole arrays or records can be added by adding the main index. The display format (hexadecimal, decimal, binary or ASCII) can be configured at the combobox in front of the value. It is ignored by string values.

The settings of the process image can be saved with the complete CANopen Device Monitor project via "File → Project → Save Project". When the project is opened again, the process image settings are restored.

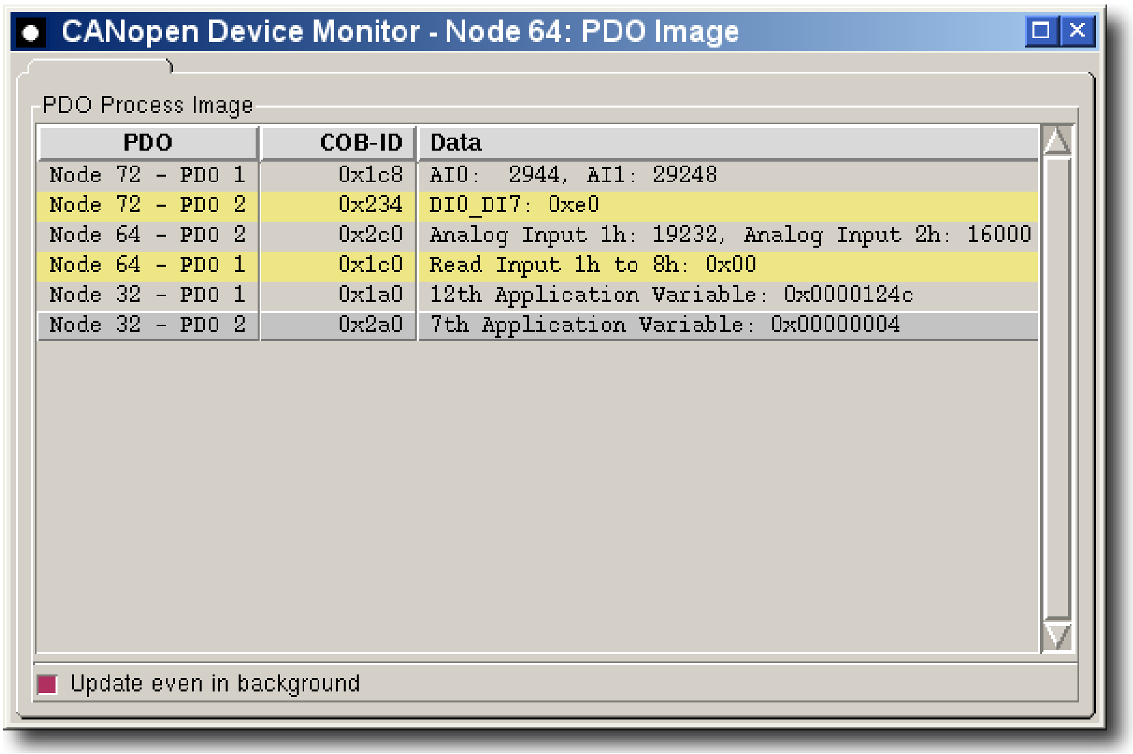
15. PDO Process Tab

The PDO Process Image tab displays the data of TPDOs from the CANopen network.

There are 2 ways to add Transmit PDOs of the CANopen nodes to the PDO process image:

- Drag&Drop the PDO object from the object tree into the PDO process image
- via the button "Add to PDO Image" in the PDO configuration tab.
  To delete all PDOs press the right mouse button and select Clear PDO Process Image This also resets the PDO settings in the CANopen server(m4d). It is sometimes required to reset the PDO settings directly in the CANopen server if it has not been started from the CDM. .

The display of the data is updated every 1000 ms when the PDO process image tab is active. To update the values even when the tab is not visible, activate *Update even in background* .



CANopen Device Monitor - Node 64: PDO Image

PDO Process Image

PDO	COB-ID	Data
Node 72 - PDO 1	0x1c8	AI0: 2944, AI1: 29248
Node 72 - PDO 2	0x234	DI0_DI7: 0xe0
Node 64 - PDO 2	0x2c0	Analog Input 1h: 19232, Analog Input 2h: 16000
Node 64 - PDO 1	0x1c0	Read Input 1h to 8h: 0x00
Node 32 - PDO 1	0x1a0	12th Application Variable: 0x0000124c
Node 32 - PDO 2	0x2a0	7th Application Variable: 0x00000004

☑ Update even in background

Figure 24: PDO Process image

The settings of the process image can be saved with the complete CANopen Device Monitor project via File → Project → Save Project When the project is opened again, the process image settings are restored.

16. Using Stripcharts

For data visualization the pre-installed **Chart** tab can be used. It shows as an example the usage of a strip-chart.

After opening the **Chart** tab, a strip-chart is displayed. Besides for the visualization of PDO data, it can be also used by scripts. Values are given to it by calling the Tcl-procedure `::cdm::addChartData`. The procedure is defined as follows:

```
proc ::cdm::addChartData { valueList } {
    # add new values to the strip-chart window
}
```

`valueList` contains a list of values for the strip-chart:

```
(EDS) 9 % ::cdm::addChartData {1 2 3 }
(EDS) 10 % ::cdm::addChartData {-1 -2 -3}
```

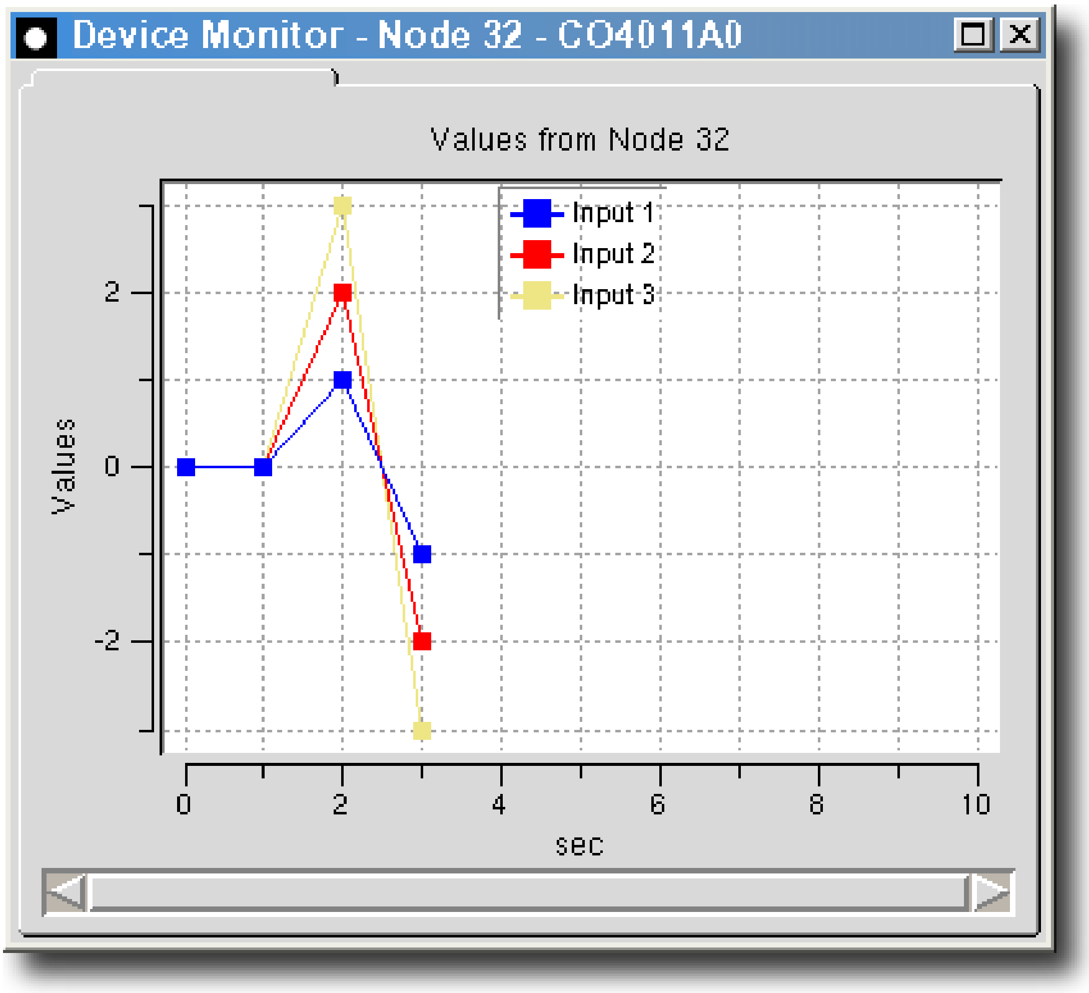


Figure 26: strip-chart after two calls to `::cdm::addChartData`

With little effort you can write your own simple script that reads values from a device and displays them in the chart.

```
% proc readAndShow { index sub dataType } {
    set value [r $index $sub $dataType]
    cdm::addChartData [list $value 0 0 0]
}
%
% # Call this procedure every second
% ::common::every "readAndShow 0x6401 0x01 i16" 1000
%
```

Windows -> Clear All , `::cdm::clearChart` or F7 clears the chart and the <space> key pauses the display.

The console or script command `::cdm::saveChart <fileName>` saves the content of the chart to a postscript file.

The chart can be zoomed by the right resp. left mouse button.

The context menu provides the following actions:

Menu entry	**Description**
Clear Chart	Deletes all values from the chart.
Save Chart	Saves the current picture as PostScript file.
Save Data	Saves all vales as CSV file.
Remove PDO	Stops PDO reception and adding of further values.

17. Extended object configuration

17.1. Slider

Optionally the tabbed field Action can be extended by further elements. A possible element is a slider (Figure 27).

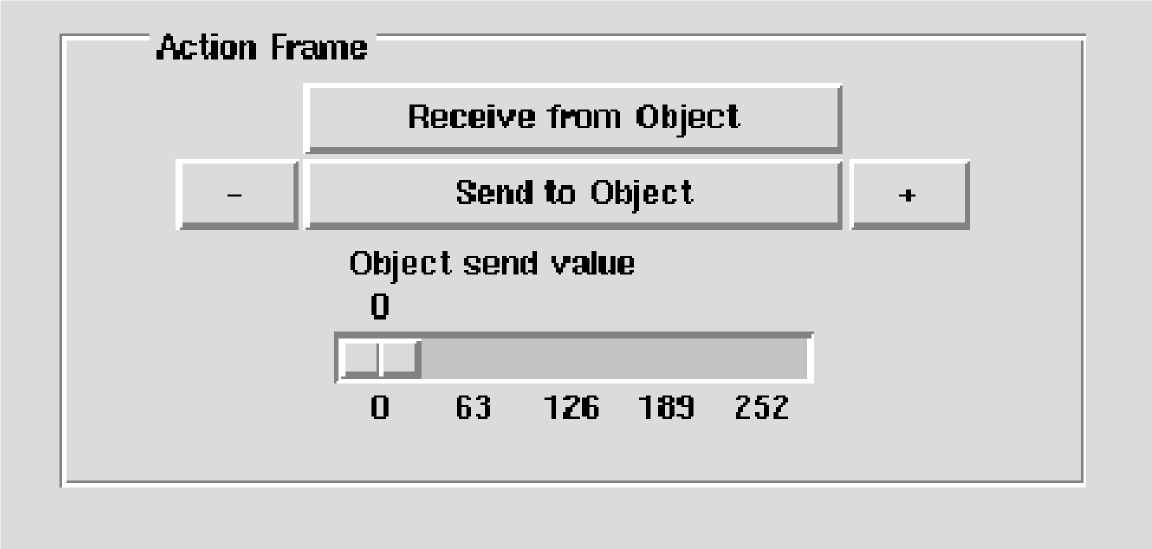


Figure 27, Tabbed field Action with slider

A slider consists of a regulator and a scale. The regulator is moving. For operating use the following keys:

- If the left mouse button is pressed in the trough, the scale’s value will be incremented or decremented by the value of the resolution option so that the slider moves in the direction of the cursor. If the button is held down, the action is repeated.
- If the left button is pressed over the slider, the slider can be dragged with the mouse.
- If the left button is pressed in the trough with the control key down, the slider moves all the way to the end of its range, in the direction towards the mouse cursor.
- The ’up’ and ’left’ cursor keys move the slider by one to left.
- The ’down’ and ’right’ cursor keys move the slider by one to right.

Further a storage of the object values with the option *used for saving configuration* is possible.

17.2. Slider in a Top Level Window

A slider in a top level window can be assigned to an object. This window stays at the desktop even if other objects are selected in the tree structure.

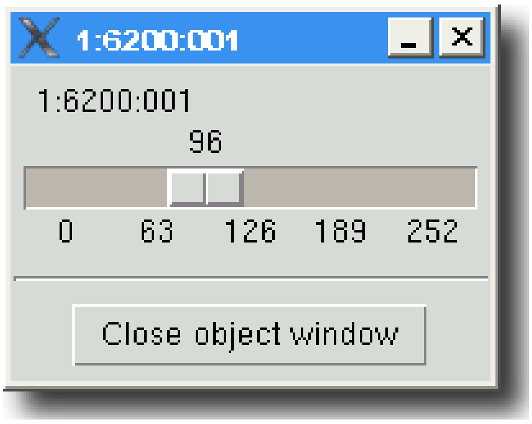


Figure 28: Slider in a top level window

The slider actions are the same as described for the Action tab slider above.

Close object window

The top level window can be closed by selecting the button labeled with "Close object window"

17.3. Switch Box Display of Objects

Each numerical object can be displayed bitwise in a unique window.

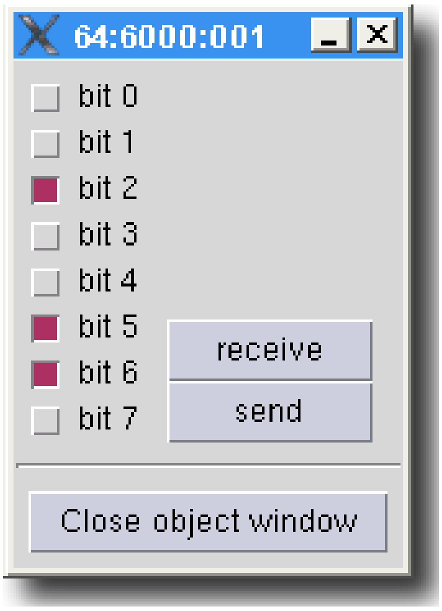


Figure 29: Bit box in a top level window

All object extension can be activated by pressing the right mouse button in the object tree.

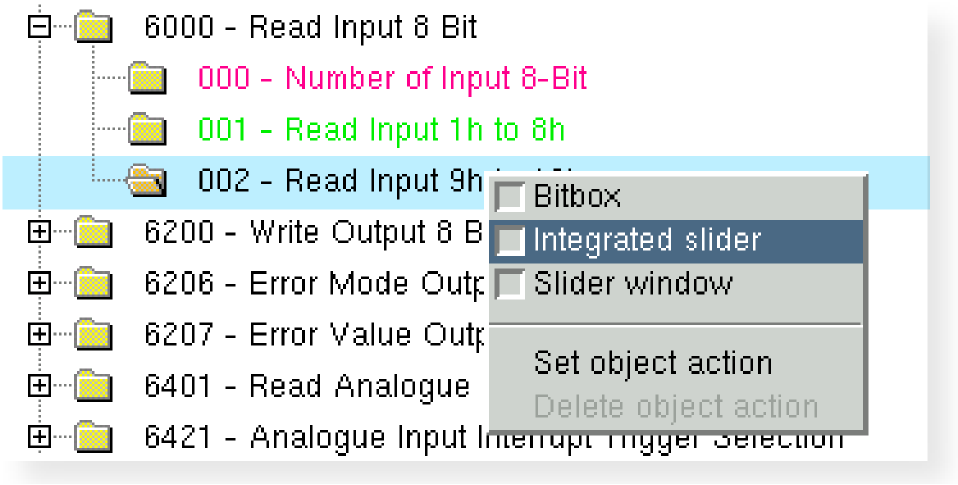


Figure 30: Configuration of the GUI Extension

The assignment of slider and bit boxes to particular objects can be saved in a OCF file.

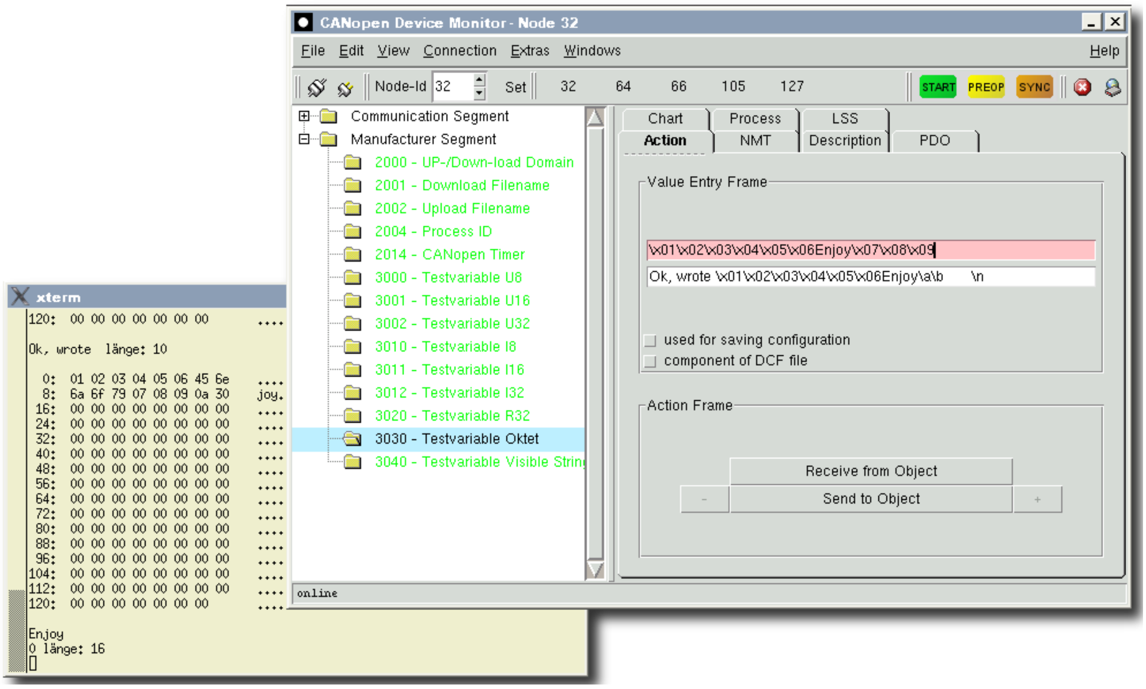
18. Usage of Octet strings

18.1. Value entry frame

In the value entry frame octet strings can be written to or read from the device. To write an octet string to a device select an object with data type octet string. Then two entries will appear in the value entry frame. The octet string can be typed into the upper entry. To specify non-printable values use the `\x00` notation. It must be *lower-case* letters and it requires exactly 2 digits after the `\x`. ASCII characters can be used as well.

EXAMPLE:
\x00\x01\x02\x03\x05\x06\x07\x08\x09\x0a\xff\xfeHello World\x0a\x00

Hit "RETURN" to send the octet string to the device. Please note that the size of an octet string is limited to 127 characters.



```
\x01\x02\x03\x04\x05\x06Enjoy\x07\x08\x09
Ok, wrote \x01\x02\x03\x04\x05\x06Enjoy\a\b      \n
```

□ used for saving configuration
□ component of DCF file

```
120:  00 00 00 00 00 00 00          ....

Ok, wrote  länge: 10

  0:  01 02 03 04 05 06 45 6e       ....
  8:  6a 6f 79 07 08 09 0a 30       joy.
 16:  00 00 00 00 00 00 00 00       ....
 24:  00 00 00 00 00 00 00 00       ....
 32:  00 00 00 00 00 00 00 00       ....
 40:  00 00 00 00 00 00 00 00       ....
 48:  00 00 00 00 00 00 00 00       ....
 56:  00 00 00 00 00 00 00 00       ....
 64:  00 00 00 00 00 00 00 00       ....
 72:  00 00 00 00 00 00 00 00       ....
 80:  00 00 00 00 00 00 00 00       ....
 88:  00 00 00 00 00 00 00 00       ....
 96:  00 00 00 00 00 00 00 00       ....
104:  00 00 00 00 00 00 00 00       ....
112:  00 00 00 00 00 00 00 00       ....
120:  00 00 00 00 00 00 00          ....

Enjoy
0 länge: 16
```

Figure 31: Octet strings

If an octet string is read from the device, it's displayed twice. The upper entry shows the value in `\x00` notation and below it is displayed as ASCII values.

18.2. Octet strings in scripts

There are two ways to send or receive octet strings from scripts. The first one is to use the normal `r` or `w` commands. In this case the data must be specified as base64.

EXAMPLE:
w 0x2000 0 os "SSBsaWtlIENBTm9wZW4gRGV2aWNlIE1vbml0b3I="

To decode or encode base64 data, the built-in tcl commands:

- `::base64::encode <raw_data>`
- and `::base64::decode <base64_data>`
can be used.

  EXAMPLE:
  w 0x2000 0 os [::base64::encode "any data"]

The other way is to use the 2 special commands:

- `::m4d::ro <index> <sub> <timeout in ms>` to read octet strings
- and `::m4d::wo <index> <sub> <timeout in ms> <data>` to send octet strings.

  EXAMPLE:
  ::m4d::wo 0x2000 0 1000 "start engine"

19. User Specified Tabs

In addition to the predefined tabbed fields (or "tabs") at the right side it is possible that you add your own tabbed fields to CANopen Device Monitor.

There are 2 kinds of tabs available:

- User specified tabs with support of tests
- User specified tabs without contents

19.1. User specified tabs with support of test

One of an additional tab is also predefined but only installed on request. It is used to organize test scripts to be controlled by the CANopen Device Monitor. It provides lists of check-boxes for the user defined test scripts.

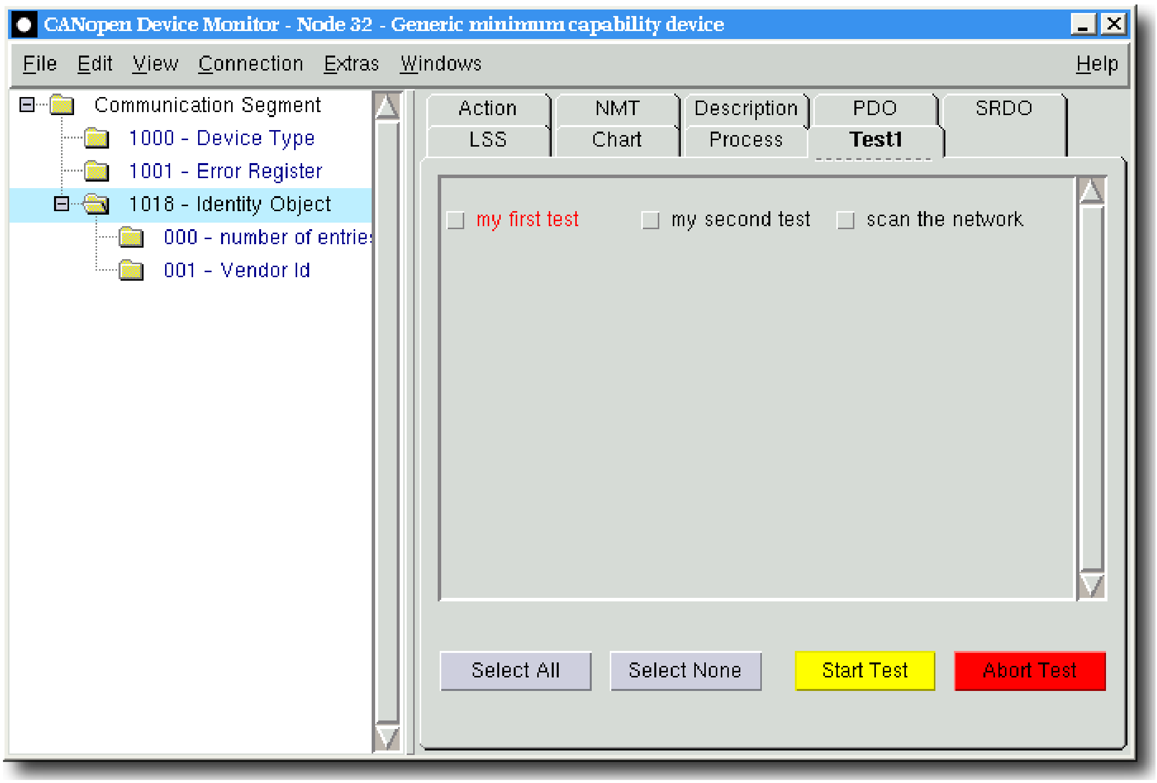


Figure 26: User Defined Tabbed Field with Test Scripts

An example can be found in the directory "<working directory>\demo_t\".

Configuration of the predefined tab for test organization is done by configuration files

- "<working directory>\demo_t\t_start.tcl" and
- "<working directory>\demo_t\t_<conf1...n>.tcl".

The file "<working directory>\demo_t\t_start.tcl" configures all preconditions for testing. It also defines all additional tab using the command `cdm::addTestTab`. Figure 26 shows an example for an additional tab.

`cdm::addTestTab t_conf {<tab_name>}`

Description

Creates an additional tab on the right side of the CANopen Device Monitor. It is predefined for organizing test scripts. It contains a widget with several check-boxes for selecting test scripts. Number and names of the check-boxes are read from the configuration file "<working directory>\<t_conf1..n>". Additional buttons are available for starting, stopping and global selection of test scripts.
If *tab_name* is not given, the name is built as "Test#" counting from 1 for each new tab.

Parameters:

t_conf	name of the file containing the names of the test scripts defaults to `t_files`
tab_name	name for labeling the tab

Return:

internal name of the tab

Format of the <working directory>\t_files:
Lines beginning with ’#’ are comments and are ignored. Each line describes one additional check-box:

```
<script>.tcl       {<label>}   {Tcl procedure name}
```

script.tcl	Name of the Tcl script that is to be executed
label	Name label for the check-box
Tcl procedure name	Tcl procedure that carries out the test. If the test was completed successfully the procedure shall return ’0’. Any other return value means an error occured. If the procedure has a parameter the test description from the `t_files` is passed in this parameter.

Script name, label and Tcl procedure are separated by space or tab.

In addition special Tcl procedures can be registered for Start of a test run, Abort of a test run, Error of a test and End of a test run. The Tcl procedures are specified as follows and are only executed for the test tab they were specified for:

```
@start <Tcl Procedure>
@abort <Tcl Procedure>
@error <Tcl Procedure>
@end   <Tcl Procedure>
```

EXAMPLE:

Two additional tab are created. The first is named with the default name "Test" the second one gets the name "Some Test". Both tabs are assigned different example scripts. Figure 26 shows the result.

t_start.tcl:

```
# add the test selection frames
# use standard values for the file (t_files) and name (Test)
cdm::addTestTab
# use user-defined values for the file (t_files2) and name (Some Tests)
cdm::addTestTab t_files2 {Some Tests}

proc my_startHandler { args } {

    cdm::banner
}

proc my_stopHandler { args } {

    puts "\n\n\n\n------------------------"
    puts "     date / sign"
    puts "\n================= E N D ================================\n"
}
```

t_files:

```
# the first set of test scripts using the standard filename
#
@start my_startHandler
@stop  my_stopHandler

t_myfirst.tcl   {my first test}   my_1st_test
t_mysec.tcl     {my second test}  my_2nd_test
# scanning the network for CANopen devices
t_scan.tcl      {scan the network} scan_test
```

t_files2:

```
# all testfiles for the second set of tests
# comments allowed after #
#
# Here we do not specify start and stop handlers
#   so they won't be called.
t_scan.tcl      {scan the network}  scan_test
t_myfirst.tcl   {my first test}     my_1st_test
t_mysec.tcl     {my second test}    my_2nd_test
t_mythird.tcl   {another}           my_3rd_test
```

t_myfirst.tcl

```
#
# Write Heartbeat Producer and check if value can be read back.
#
#
proc my_1st_test { args } {

    wwc 0x1017 0 u16 1000 OK
    if { $::global_stop == "1" } {
        return "Canceled
    }
    rrc 0x1017 0 u16 1000

    return 0
}
```

The tabs are visible after loading the file with the tab specification. Loading can be done via the menu File → Load File → t_start.tcl or by issuing the `source` command in the Console:

```
$ source t_start.tcl
```

Once after loading the new tab, any changes at the files "<working directory>\t_start.tcl" and "<working directory>\t_<t_conf1...n> are only recognized and valid after a restart of the CANopen Device Monitor and reloading of "<working directory>\t_start.tcl".

The tabs for testing contain the following additional control buttons:

Select All
: mark all scripts for execution

Select None
: deselect all check-boxes, remove all scripts from execution

Start Test
: start loading and executing of the selected test scripts

Abort Test
: stop execution of test scripts
Normally the execution stops at the end of the currently running script. If a running script has to abort immediately, it must do some preperations. The script must look for the state of the global variable *global_stop*. If the "Abort Test" button is selected the value of *global_stop* is set to 1. The script on the other side must not block the User Interface event loop so that the user is be able to select the Stop button. Therfore the Tcl function `update` must be called regularly.

EXAMPLE:

Template for testing *global_stop* in test scripts

```
# global variables
global global_stop

# initialize global variables
set global_stop ""

# execute application
while { ($global_stop == "") } {
    .
    .
    # read actual value of global_stop
    # and update the GUI
    update
}
```

19.2. User specified tabs without contents

A new tab will created by the following command:

```
cdm::addTab {<title> <pos>}
```

Description

creates an additional empty tab

Parameters:

title	name of the tab
pos	position of the tab in the display default: append as last tab

Return:

internal name of the tab

The empty tab can be designed by the application by input of Tcl commands at the Console or by Tcl scripts.

EXAMPLE:

```
# Create an empty tab at position 0 with the title "Service".
set name [cdm::addTab Service 0]

# Create a button in the center of the bottom of the tab
# with the name "OK".
button $name.button -text "OK" -command {w 0x1017 0 u16 500}
pack $name.button -side bottom -anchor center
```

19.3. Erasing User Specified Tab-sets

Tab-set can be deleted with the command:

```
cdm::deleteTab {<pos>}
```

Description

deletes an additional tab

Parameters:

pos	position of the tab in the display (starting with 0)

Return:

nothing

20. Data Management with DCF File

The CANopen Device Monitor can handle object values as DCF files (ASCII format) according to CiA-306 as XDC files (XML format) according to CiA-311 with the following features:

object code		
	VAR	yes
	ARRAY	yes
	RECORD	yes
	DOMAIN	no
compact storage		
	for PDOs	analog to EDS
	for Arrays	no
Denotation		no

The base of the DCF file handling is the EDS file, therefore it is not possible to load or save a DCF file without loading of EDS file before. The handling of DCF files is also not possible when the object dictionary is scanned from the device.

Only the `ObjectList` is support when using XDC files.

20.1. Creating a DCF file

The file name of the DCF file can be selected freely by the user.

Objects which are relevant for the DCF file have to be marked by the checkbutton "component of DCF file" on the tab field Actions.

The CANopen Device Monitor communicates via SDO with the device by the usage of an internal data buffer.

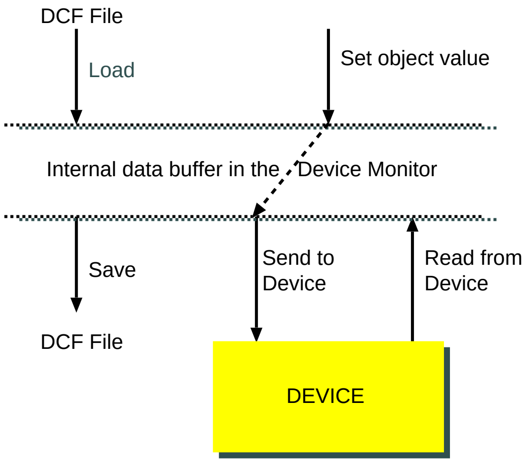


Figure 33: DCF data flow

20.2. Load

DCF files can be loaded via File → Load device configuration → Load DCF

Before a DCF file will be loaded, the CANopen Device Monitor executes a consistency check. The DCF file has to match to the EDS file. Criteria for the consistency check are:

- The EDS file name entered in the DCF file (LastEDS).
- The date and time of creation/modification in the EDS and DCF file. To guarantee the consistency of the object dictionary the EDS file has to be older than the DCF file.

Loading of a DCF file does not change the CAN bit rate and the node-id in the CANopen Device Monitor. The parameter values are only loaded to the internal data buffer of the CANopen Device Monitor and are transfered to the device after Send (see chapter "Data Management with DCF file/Send").

20.3. Send

All selected parameter values will be sent from the internal data buffer of the CANopen Device Monitor to the device per SDO via Extras → Send conf to Device Sending is also allowed when the parameter values are not stored in a DCF file.

21. Offline Mode

The SDO communication to the device can be switched on/off with the options offline/online in the menu Connection when the Device Monitor is connected. Therewith it is possible to create a DCF file without communication with the device.

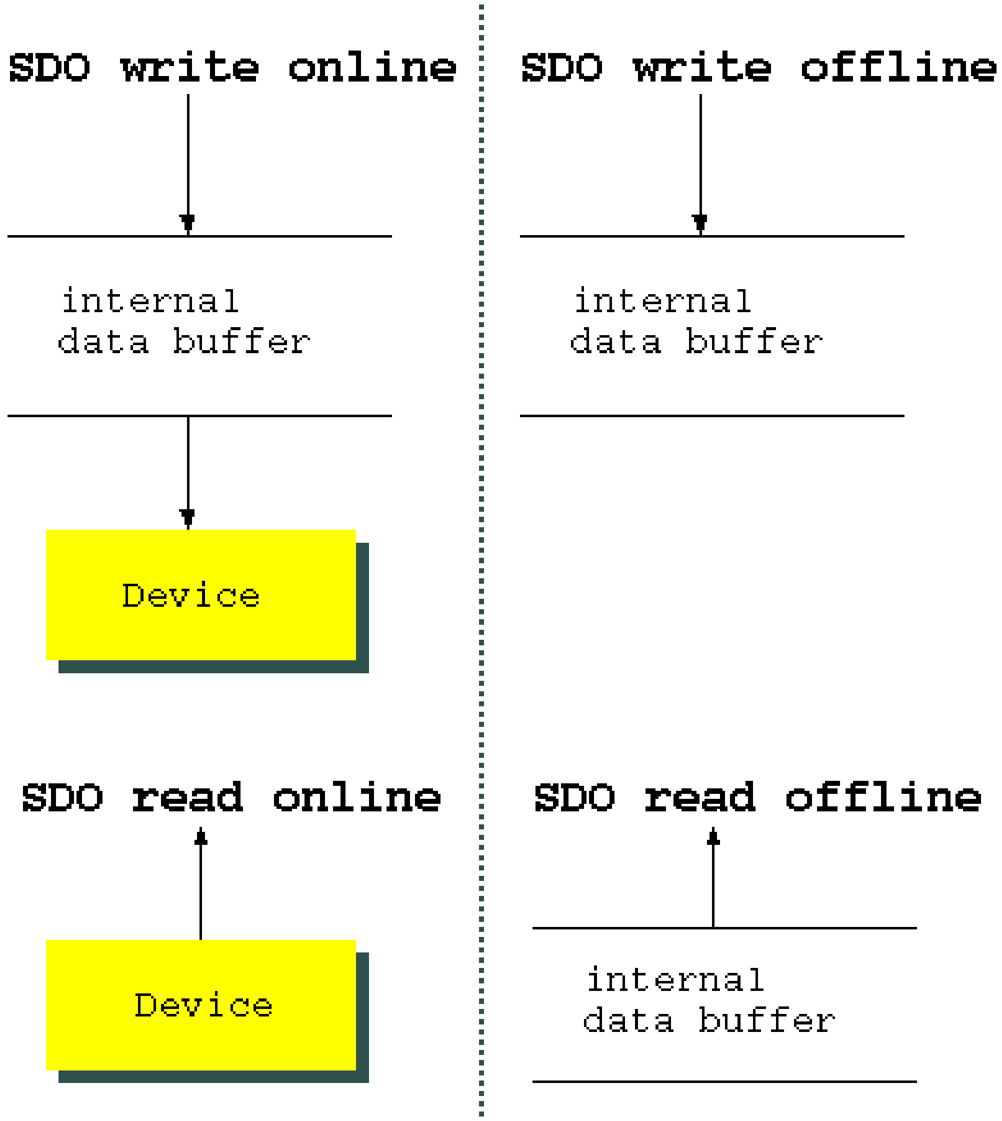


Figure 34: SDO communication depending on the options offline/online

22. Object Data Management with OCF File

22.1. Saving of OCF files

The values of configured objects of CANopen devices and additionally settings of bit boxes, slides and object specific commands can be stored in a file for restoring them automatically in another session.

The objects to be stored are selected by an active check-box labeled with "used for saving configuration" at the tabbed field Action.

The configuration can be stored by the menu: File → Save device configuration → Save OCF . Each value is obtained from the CANopen device in the moment it is stored. The command to save object values results in CAN traffic with SDO transfers.

The CANopen Device Monitor does not automatically set the file extension.

In difference to the DCF file the OCF file needs less memory, because the OCF file only contains the selected parameter values.

22.2. Loading

Saved object values stored in a configuration file can be reloaded to the CANopen device with the menu File → Load device configuration → Load OCF . The values are transfered from of the configuration file to the CANopen device which result in CAN traffic by SDO transfers.

22.3. File Format

The first lines contain some header information for maintaining the data, like CANopen Device Monitor version, time and date. The device configuration file contains commands for the CANopen Device Monitor for writing values to a device. Additionally the configuration of bit boxes and sliders is saved in the OCF-file.

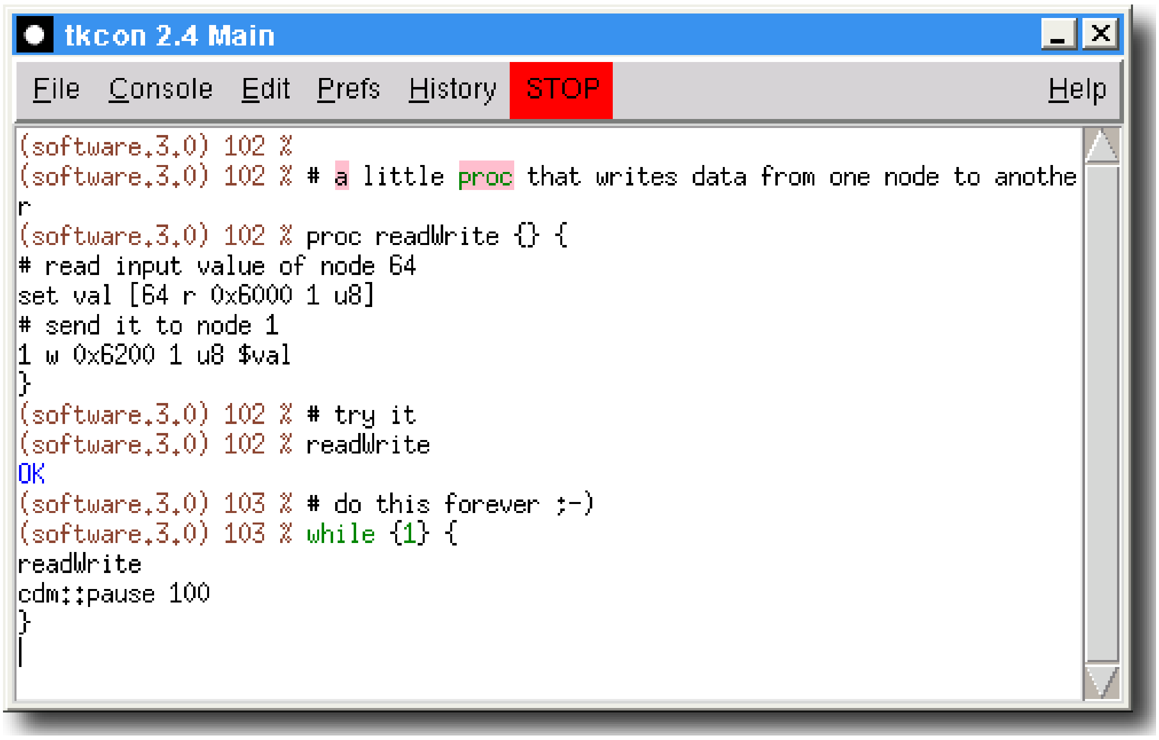
EXAMPLE:

```
# OCF file for: SPC Operating Interface
# created by CANopen Device Monitor V3.2.7
# Wed Mar 30 14:48:40 (CEST) 2005
w 0x100C 0 u16 100
w 0x2000 0 u8 67
::cdm::setSlider {} 100D 000 "0 255 1"
```

23. Console

The Console can be activated via View → Console

In the console Tcl commands can be executed as well as user scripts and procedures.



```
(software.3.0) 102 %
(software.3.0) 102 % # a little proc that writes data from one node to anothe
r
(software.3.0) 102 % proc readWrite {} {
# read input value of node 64
set val [64 r 0x6000 1 u8]
# send it to node 1
1 w 0x6200 1 u8 $val
}
(software.3.0) 102 % # try it
(software.3.0) 102 % readWrite
OK
(software.3.0) 103 % # do this forever ;-)
(software.3.0) 103 % while {1} {
readWrite
cdm::pause 100
}
```

Figure 35: Console

23.1. Tcl Commands

A description of the Tcl script language exceeds the scope of this manual. To illustrate some special features and basics some simple examples will be given for using Tcl commands in the Console window.

The bibliography refers to books and web pages for the Tcl language.

EXAMPLE:

```
set val 5        ;#set the variable val to the value 5
set val          ;#show the current value of the variable val
puts "Hello"     ;#put the word Hello at the Console
# comment
set myarray(baud)   19200      ;#define the array myarray
set myarray(parity) even
# variables are referenced by using its name preceeded with "$"
puts "Bitrate: $myarray(baud)" ;#reference of the array myarray
```

Further information can be found at www.tcl.tk ⟨http://www.tcl.tk⟩.

23.2. Scripts

Sequences of Tcl commands inclusive of controlling structures can be created with a text editor. These files used to have the extension *.tcl and can be loaded from the CANopen Device Monitor:

per menu:	File → Load File
interactively:	input in the Console `source <file>.tcl`

Command sequences can be combined to procedures. The procedure is executed by calling the name of the procedure.

Procedures saved in Tcl files are available after loading the script file.

EXAMPLE:

file example.tcl:

```
# --------------------------------------------------------------
# show Hello
# --------------------------------------------------------------
proc showHello { name } {
    puts "Hello $name"
    puts "How are you?"
    return
}
```

load the script with the source command in the Console:

```
source example.tcl
```

execute the defined procedure in the Console interactively:

```
$ showHello Heinz
Hello Heinz
How are you?
```

24. Scripting with PDOs

To test the PDO handling of a device or to simulate complex control processes, PDOs can be configured, sent, received and requested by scripts and by console commands.

! After a modification of the PDO configuration by Tcl/Tk-Scripts the reception and the transmission of PDOs in the PDO mask is not supported anymore.

As the CANopen Device Monitor resp. the CANopen server acts like a CANopen device within the network, PDOs must be configured internally, so that they can be sent or received.

24.1. Configuration of PDOs in scripts

To configure the PDOs the point of view of the remote device is used. PDOs, sent from the device are TPDOs and PDOs, that are received by the device are RPDOs. To configure the PDOs the commands `::pdo::set_tpdo` und `::pdo::set_rpdo` can be used. The following example shows the configuration of a TPDO. For RPDOs it is simular and described in the section "PDO commands".

EXAMPLE:

The device has one TPDO (TPDO No. 1), which contains the objects 0x6000:1 and 0x6000:2. It is transmitted event-trigged by the device. Its COB-ID is 0x221.

The appropriate call to `::pdo::set_tpdo` is:

```
::pdo::set_tpdo local 1 0x00000221 event 0x6000 1 0x6000 2
```

The parameter ’local’ means, that only the CANopen Device Monitor shall be configured. If it were ’remote’, the remote device would be configured accordingly as well using SDOs.

The **1** is the number of the PDO. It is of importance for further commands. After that follows the COB-ID and the transmission type.

The last parameters are the mapped objects. They have to be specified as pairs of index and sub index. Please regard that the EDS file for the device must be loaded and that the objects must exist in the EDS file.

To change an already configured PDO, it must be deactivated before. To deactivate the PDO, the MSB of the COB-ID must be set to 1. The COB-ID to deactivate the PDO in the example above is 0x80000221.

24.2. Transmission of PDOs

To send a PDO from the CANopen Device Monitor to the device, it must be configured previously with `::pdo::set_rpdo`, because it is a RPDO of the device. After that it can be sent by `::pdo::wpdo` to the device.

EXAMPLE:
The RPDO 2 of the device shall receive two UNSIGNED16 values and it is already configured.
::pdo::wpdo 2 2 0x1234 0xfedc
The 1st 2 is the number of the PDO. The 2nd 2 is the length of the PDO, i.e. the number of mapped objects.

Take into account that PDOs only can be sent or received in the state OPERATIONAL.

24.3. Request of PDOs using RTR

If RTR is supported, configured TPDOs of the device can be requested by RTR. Therefore it must be configured with `::pdo::set_tpdo`. The following example shows how to request PDO 1.

EXAMPLE:
::pdo::rpdo 1

24.4. Reception of PDOs

To receive a PDO it must be configured (`::pdo::set_tpdo`) and an asynchronous indication function must be present.

The following indication function simply writes the data to the console.

```
# putPDO --
#  puts PDO data to the console
proc putPDO { num len dataList } {
   puts "PDO: $dataList"
}
```

Any indication function must have the 3 parameters `num len dataList`.

Each PDO needs an indication function. This can be configured with the command `::pdo::setPDOIndication`.

EXAMPLE:
::pdo::setPDOIndication 1 putPDO
The first parameter is the number of the PDO and the second one is the name of the indication function.

If the indication function is not tailored for a specific PDO, more than one PDO can be assigned to it.

24.5. Waiting for PDOs

The function `::pdo::waitForPDO <num> <expr>` can wait for PDOs. <num> specifies the number of the PDO. It must be configured before it can be used. With the optional parameter `expr` a numeric expression can be specified, that must be 1 to return. Within the expression the variables `val(1)` to `val(8)` can be used to access the objects in the PDO. `expr` is evalutated by the tcl function `expr`. The following script shows an example.

```
# wait until the bit TargetReached is set
#
set script { ($val(1) & 0x40) > 1) }
waitForPDO 1 $script
```

25. CAN Message Logging

Each CAN message on the CAN bus can be displayed in a separate message log window. It can be activated by selecting View → Message Log

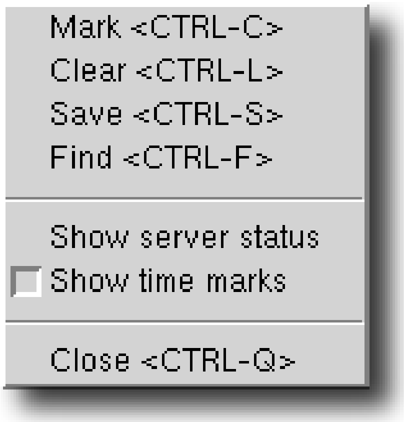


Figure 36: Message Log file menu

Menu entry	Description
Mark	Adds a marker
Save	Saves the content in CAN-REport format
Show server status	Shows status information of the CANopen server
Show time marks	Inserts a time mark every minute
Find	Opens a Find dialog
Close	Closes the log window

The message is displayed in the CAN-REport message format:

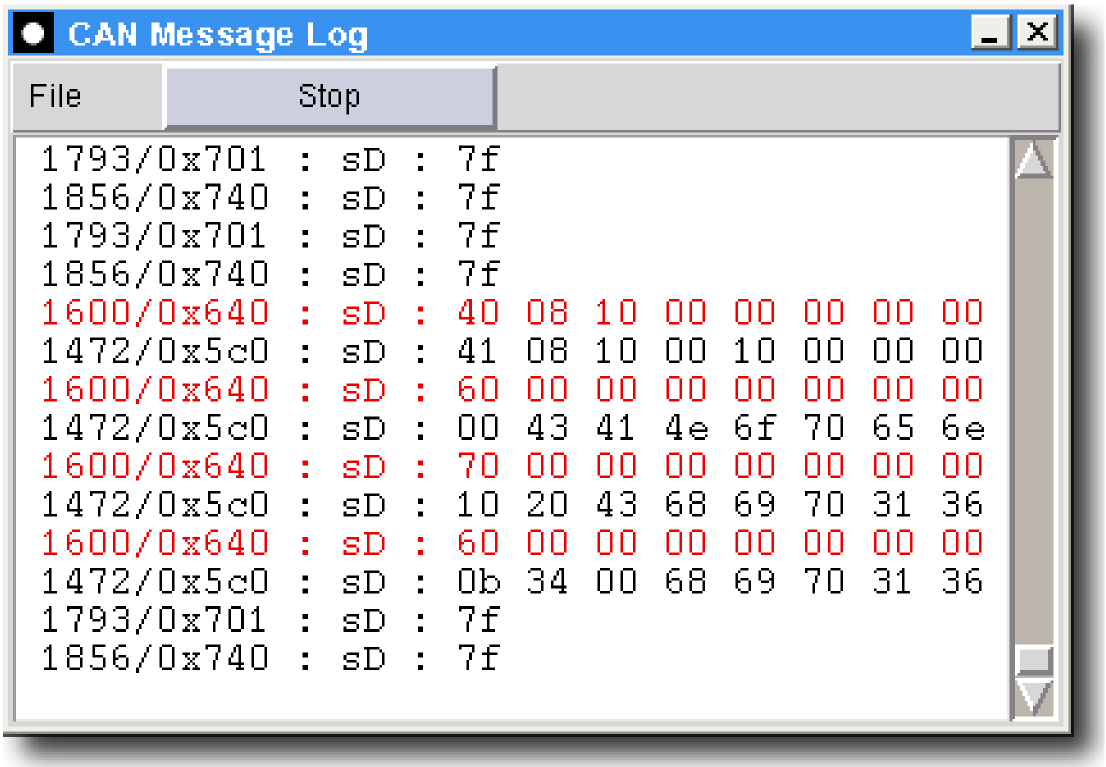
<COB-ID (dec)>/<COB-ID (hex)> : <Type> : <Data (hex)>

EXAMPLE:

```
 1000/0x3e8 : sD : 11 22 33 44 55 66 77 88
```

A data message with ID 3E8h was received.
It contains 8 data bytes 11h, 22h, 33h, 44h, 55h, 66h, 77h and 88h.

The message logs can be loaded with the CAN-REport and so they can be interpreted with its sophisticated extensions.



```
1793/0x701 : sD : 7f
1856/0x740 : sD : 7f
1793/0x701 : sD : 7f
1856/0x740 : sD : 7f
1600/0x640 : sD : 40 08 10 00 00 00 00 00
1472/0x5c0 : sD : 41 08 10 00 10 00 00 00
1600/0x640 : sD : 60 00 00 00 00 00 00 00
1472/0x5c0 : sD : 00 43 41 4e 6f 70 65 6e
1600/0x640 : sD : 70 00 00 00 00 00 00 00
1472/0x5c0 : sD : 10 20 43 68 69 70 31 36
1600/0x640 : sD : 60 00 00 00 00 00 00 00
1472/0x5c0 : sD : 0b 34 00 68 69 70 31 36
1793/0x701 : sD : 7f
1856/0x740 : sD : 7f
```

Figure 37: Message Log window

Not all CAN drivers of the CANopen Server **m4d** are able to send all received messages to CDM and not all are able to display it's own messages sent. In this case messages sent are not displayed or displayed in red color. The message shown must not be displayed in the correct time order. It is only the time when the message is scheduled to the driver. The message itself is sent when there are no messages with a higher priority on the bus.

The Message Log has a remarkable influence on the performance of the CANopen Device Monitor at high bus loads. To increase the performance close the Message Log window.

! Message logging is not supported by every CANopen server.

25.1. Scripting-Interface

Some parts of the functionality of the Message Log can be used by scripts.

The provided functions are:

`open_messagelog`

Description:

Open the message log window.

Parameters:
none

Results:
none

`clear_messagelog`

Description:

Clear the content of the message log.

Parameters:
none

Results:
none

`save_messagelog <fileName>`

Description:

Save the content of the message log into a file.

Parameters:

fileName	path to file

Results:
none

26. SDO Program Download

The Device Monitor is capable of downloading files to CANopen devices. This can be any kind of file, e.g. new software versions, parameters, etc. The CANopen device has to support the SDO Domain transfer.

Click with the right mouse button on a domain object to open the download menue. Select upload or download to start the domain transfer.

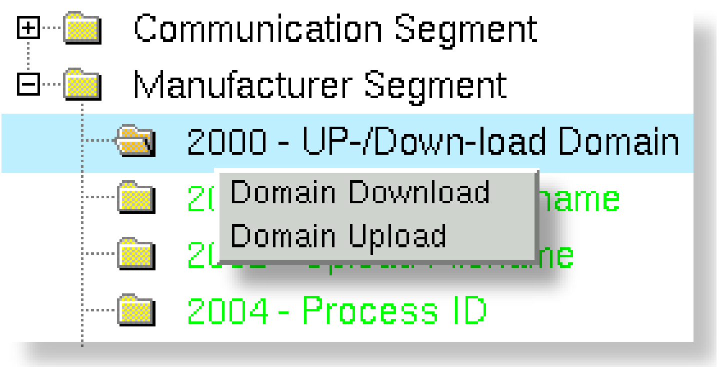


Figure 38: Domaintransfer

Download from a script or from the console is performed by calling:

```
::cdm::domainDownload  <node>  <index>  <sub>  <timeout>
<file>
```

Description

executes a program download

Parameters:

node	Node-ID
index	index of the domain object where the file shall be downloaded
sub	subindex of the domain object where the file shall be downloaded
timeout	timeout in milliseconds
file	absolute path name of the file to be downloaded

Return:

nothing

Paths to the files that contain spaces must be enclosed in quotes. Additionally the POSIX style must be used to specify the path.

EXAMPLE:

```
%
% ::cdm::domainDownload 32 0x1f50 1 25000 \
          "D:/Dokumente und Einstellungen/Administrator/Desktop/prog.bin"
%
```

A domain transfer from the device to the CANopen Device Monitor can be done by:

```
::cdm::domainUpload <node> <index> <sub> <timeout> <file>
```

27. Network overview window

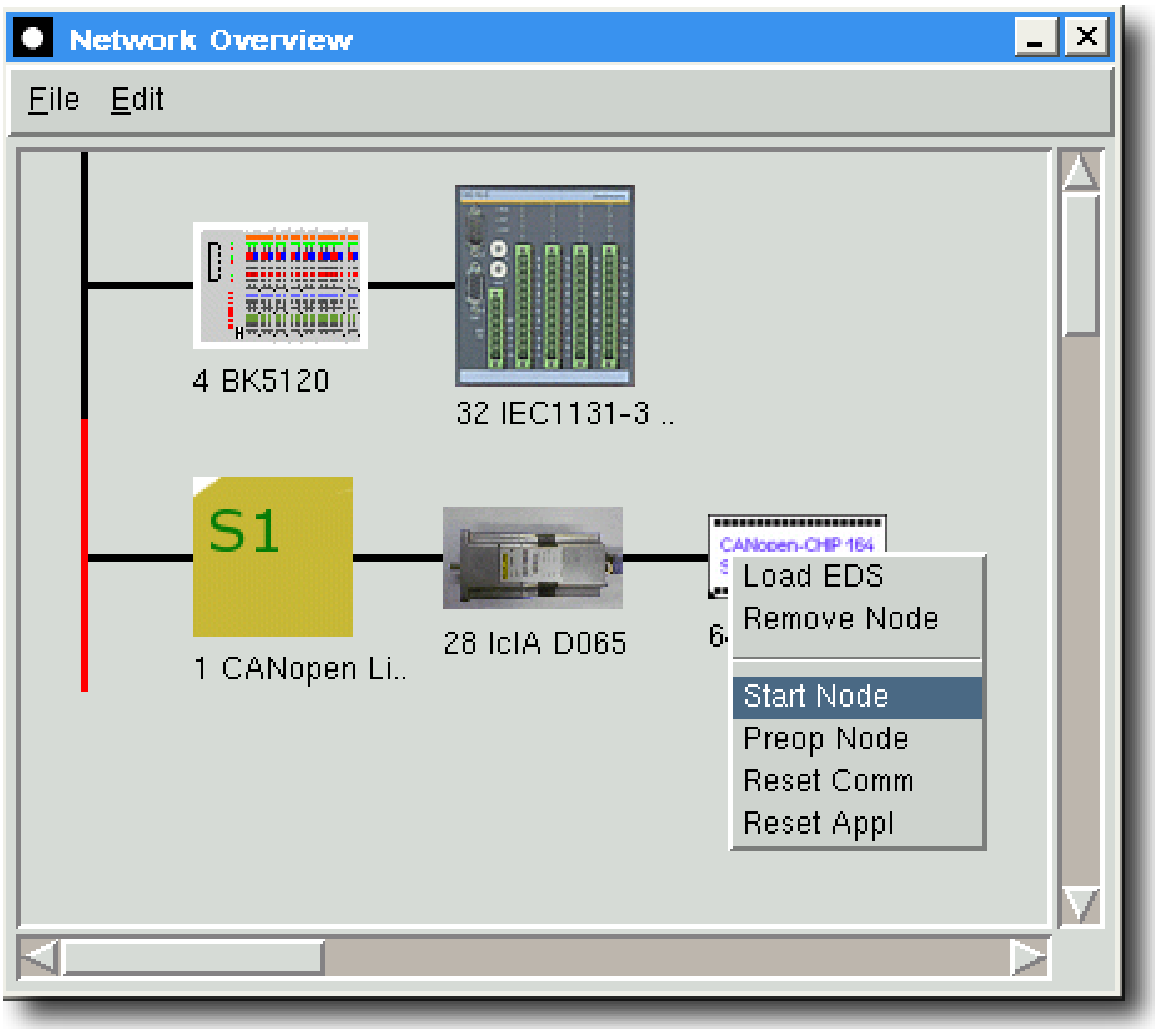


Figure 39: Network overview window

The network overview represents the nodes graphically organized in groups. Pictures in GIF format are necessary for the visualization of the devices. These picture files must have the same name as the EDS file and must be in the same directory. Their maximum size can be 72 x 72 pixel.

The node menu which provides reloading of the EDS file and access to the NMT commands can be accessed via the right mouse button.

28. Project Files

Project files are useful for projects with more than one device resp. EDS-File. The can be saved and loaded via: File -> Project and contain the following data:

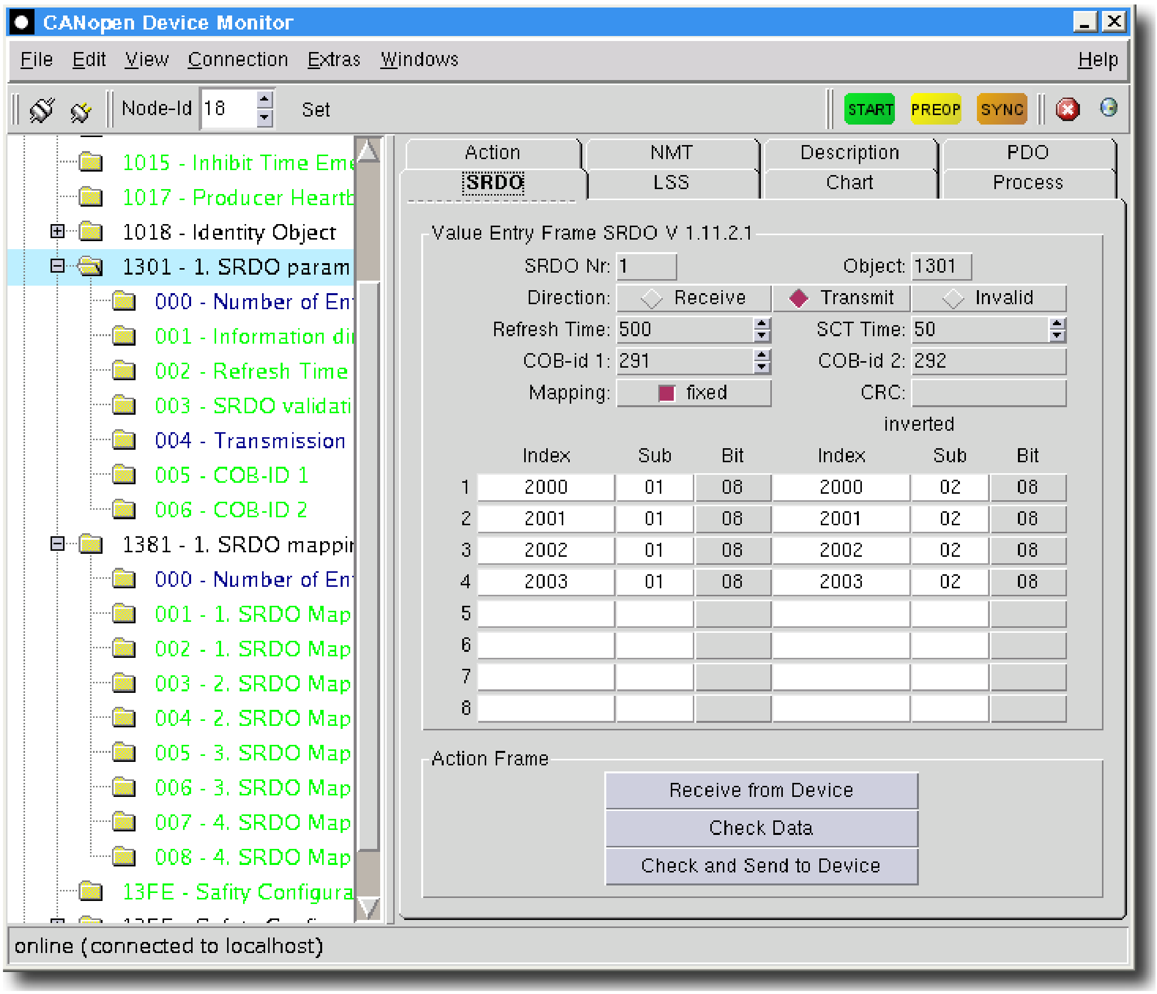
- all nodes with links to their EDS files
- links to the DCF files containing
  - □ device-specific values for the objects (ParameterValue)
- links to the OCF files containing
  - □ device-specific values for the objects
  - □ configuration settings for bit boxes, sliders and object-specific CallBack functions
- configuration of the "Process Image" tab
- configuration of the "PDO Process Image" tab

If a project is loaded all nodes and there EDS files are loaded. If DCF files are available for the nodes these are loaded too and the values in the DCF file are stored into the internal buffer. OCF stored configurations and configurations made at the "Process Image" tab are restored.

28.1. Import/Export of CCM project files

Project files of the CANopen Configuration Manager (CCM) can be loaded. Likewise the CCM can import project files of the CANopen Device Monitor. Not all settings are imported in both ways, but the Node<->EDS assignments and the connections defined in the CCM are preserved.

29. SRDO Tab



Value Entry Frame SRDO V 1.11.2.1

SRDO Nr: 1 Object: 1301

Direction: ○ Receive ● Transmit ○ Invalid

Refresh Time: 500 SCT Time: 50

COB-id 1: 291 COB-id 2: 292

Mapping: ▪ fixed CRC:

				inverted		
	Index	Sub	Bit	Index	Sub	Bit
1	2000	01	08	2000	02	08
2	2001	01	08	2001	02	08
3	2002	01	08	2002	02	08
4	2003	01	08	2003	02	08
5						
6						
7						
8						

online (connected to localhost)

Figure 40: SRDO mask

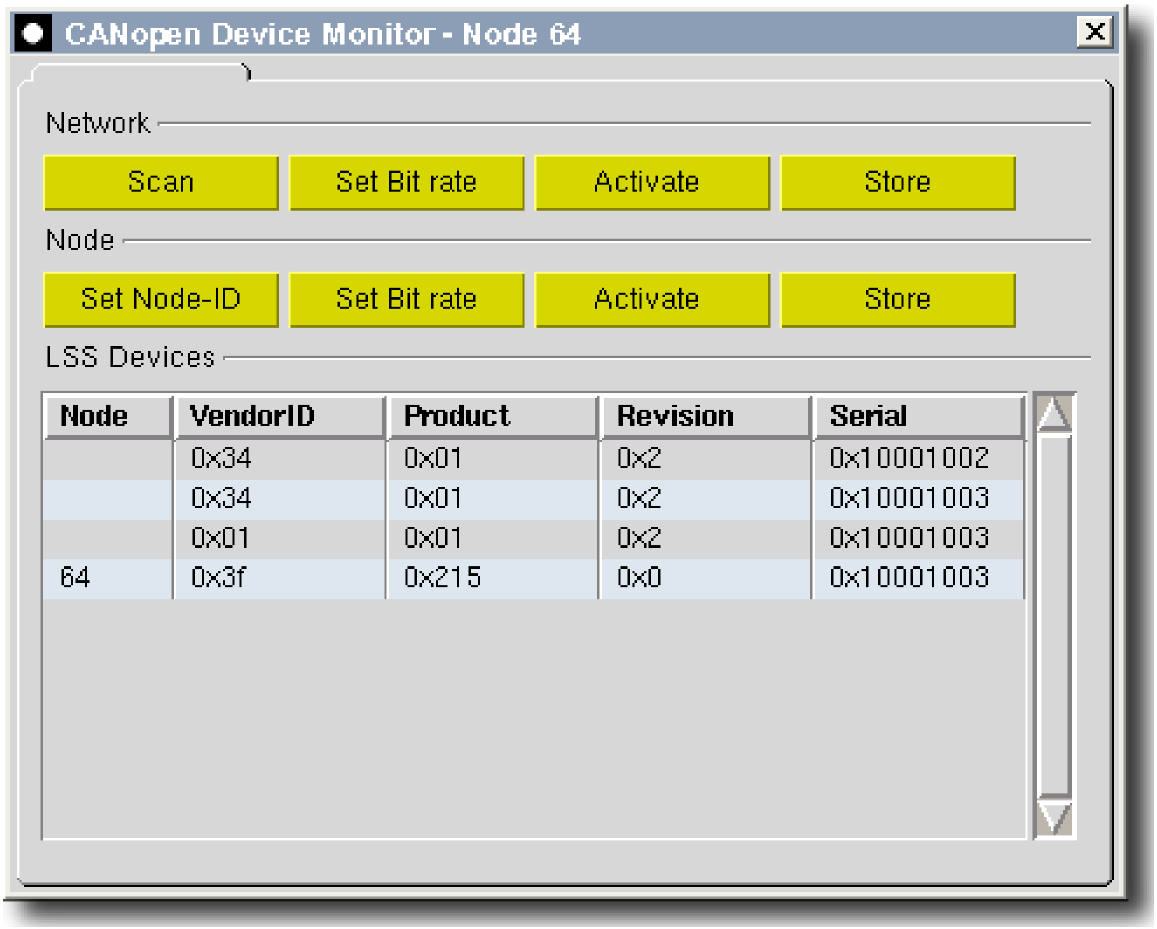
The SRDO tab combines all settings of a SRDO. At the selection of a SRDO Object in the object dictionary the SRDO tab is updated with the data from the EDS or with previous read data from the device. By pressing the "Read from Device" button the data are read immediately from the device. The upper part of the tab shows the actual SRDO data from the SRDO object like the COB-IDs. The table below shows the mapping values from the mapping object.

This tab can be activated via Extras → Plug-in -> Safety. It is only available, if the appendant license has been purchased.

Button	Function
Receive from Device	Read the data of the current SRDO object and of the appendant mapping object
Check Data	The values are checked for consistency and compliance with the standard.
Check and Send to Device	The values are checked for consistency and compliance with the standard and transmitted to the device. The mapping data are only transmitted if the mapping is not fixed.

30. LSS Tab

30.1. LSS Mask



Node	VendorID	Product	Revision	Serial
	0x34	0x01	0x2	0x10001002
	0x34	0x01	0x2	0x10001003
	0x01	0x01	0x2	0x10001003
64	0x3f	0x215	0x0	0x10001003

Figure 41: LSS Tab

The LSS tab simplifies the configuration of CANopen nodes using the "Layer Setting Services".

This tab can be activated via Extras → Plug-in -> LSS. It is only available, if the corresponding license has been purchased.

Button	Description
Scan Network	Scans the net for unconfigured devices.
Set Bit rate	Changes the bit rate of all device within the network.
Activate	Activates the changed bit rate of the device.
Store	Saves the changed bit rate in non-volatile memory.
Set Node-ID	Sets the node-ID of the device selected in the list below.
Set Bit rate	Changes the bit rate of the device selected in the list below.
Activate	Activates the changed bit rate of the device selected in the list below.

Button	Description
Store	Saves the changed bit rate in non-volatile memory.

Devices that are already configured can be added to the list by Add configured device which is available via the right mouse button.

30.2. LSS Commands

In scripts or in the console the following LSS commands are available:

Switch Selective

```
lss switch_sel <vendorId> <product> <revision> <serialNo>
```

Set single LSS slave in LSS CONFIGURATION state.

Switch global

```
lss switch_glob <0|1>
```

Set complete network in LSS CONFIGURATION (1) or LSS OPERATION (0) state.

Configuration of node id

```
lss set_node <nodeId>
```

Set the node id of an LSS slave in the state LSS CONFIGATION.

Request node

```
lss get_node
```

Get the node id of an LSS slave.

Identify LSS slaves

```
lss identity <vendorId> <product> <rev lo> <rev hi> \
                         <serial low> <serial hi>
```

Scans the network for nodes that are in the given address range.

Bitrate Configuration

```
lss set_bitrate <table_sel> <table_idx> \
                            [<gw_table_sel> <gw_table_idx>]
```

Set the new bitrate of an LSS slave. The LSS slave has to be in state LSS CONFIGURATION.

The first two parameter define the bitrate of the LSS slave. The last two parameter define the bitrate of the CANopen-Server. They are used when autobaud is to be set at the LSS slaves.

Bitrate	Table index <table_idx>
1000	0
800	1
500	2
250	3
125	4
reserved	5
50	6
20	7
10	8
Autobaud	9

Only table 0 <table_sel> the standard CANopen table is supported by the CANopen Device Monitor.

Bitrate activation

```
lss activate_bitrate <time>
```

Activates the bitrate. The CANopen Device Monitor responds after 2 * time is elapsed. The time is given in milli seconds.

Store Configuration

```
lss store
```

On reception of this command the LSS slave saves the bitrate and node id. The LSS slave has to be in LSS CONFIGURATION state.

31. DSP 402 extension

The DSP 402 extension consists of 3 components for the simplification of the work with drives in conformity with the DSP402.

This extension can be activated via Extras → Plug-in -> DSP 402 Extension. They are only available, if the corresponding license has been purchased.

31.1. State machine tab

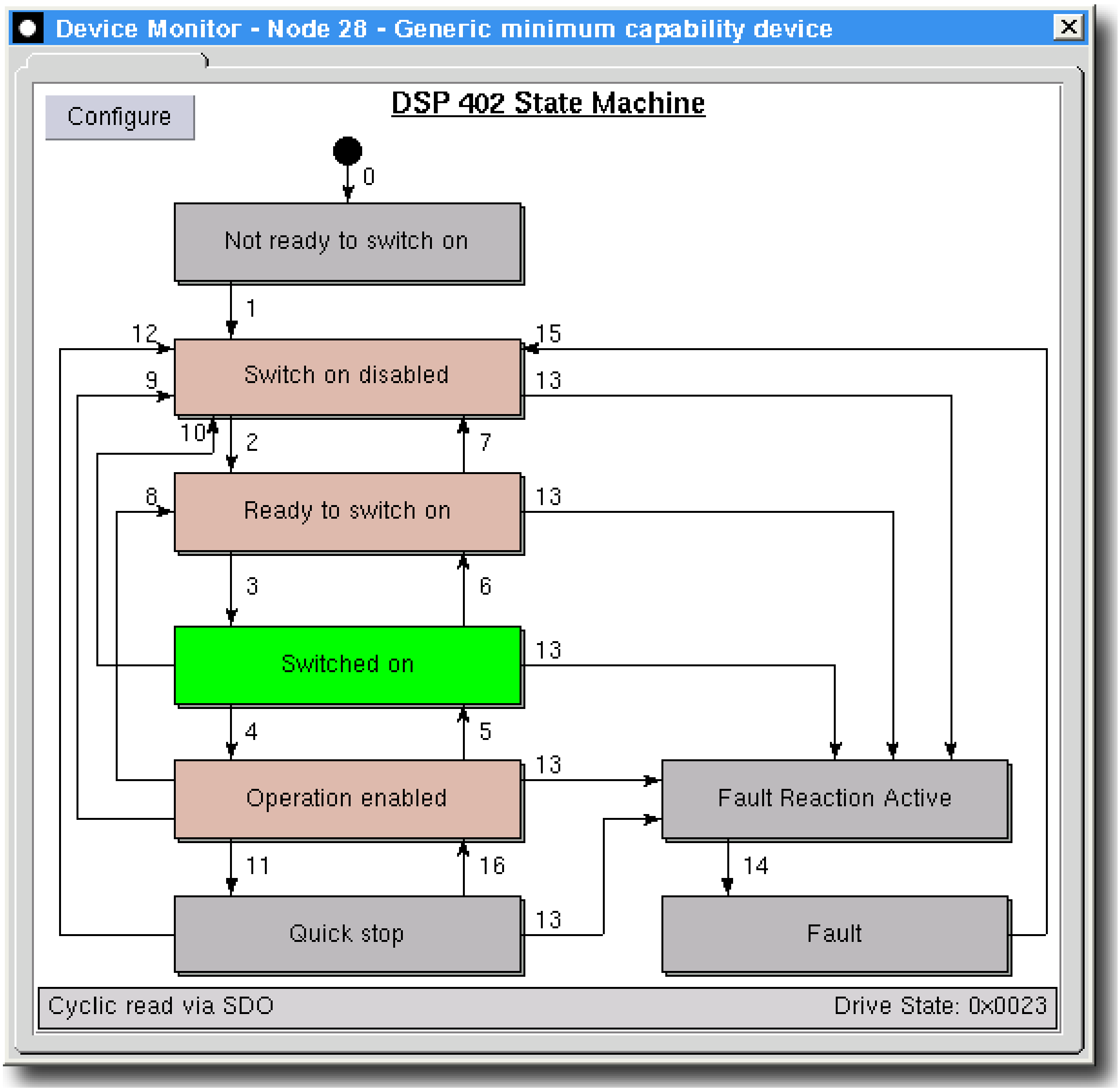


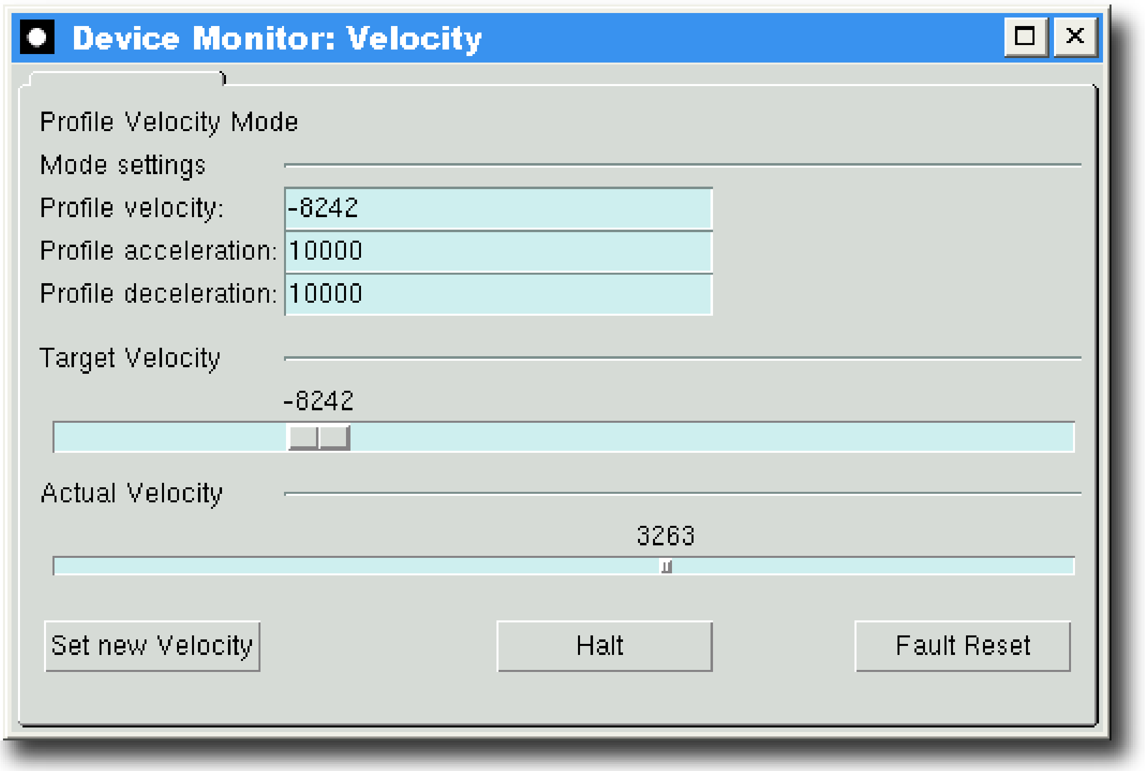
Figure 42: DSP402 state machine

The state deposited green is the current state of the drive. Pale brown fields indicate possible next states and gray fields aren't obtainable directly from the current state. The current value of the status word is displayed under the state machine down on the right.

The configuration dialog can be opened by the button "Configure". This dialog allows the configuration of the acces mode to the device and the axle of the device. These settings are also valid for the other DSP402 extensions.

31.2. Profile velocity mode tab

The profile velocity mode tab simplifies controlling CANopen drives in the profile velocity mode.



Device Monitor: Velocity

Profile Velocity Mode

Mode settings

Profile velocity: -8242

Profile acceleration: 10000

Profile deceleration: 10000

Target Velocity

-8242

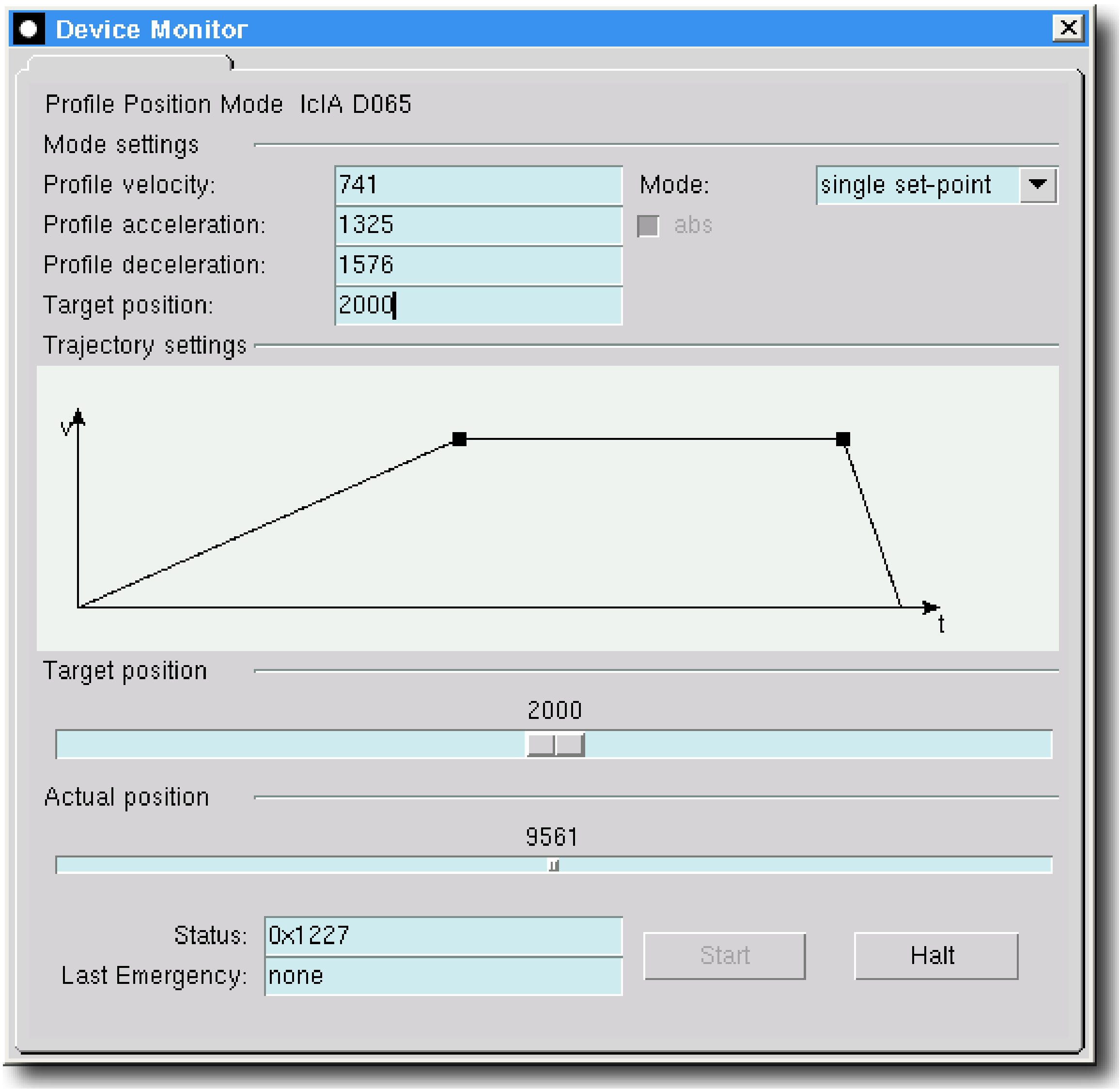
Actual Velocity

3263

Figure 43: Profile velocity mode tab

31.3. Profile position mode tab

The Position Mode Tab simplifies the test and the commisioning of a device. The speed, the acceleration and the delay as well as the target position can be configured comfortably. Limiting values for these parameters are read from the objects of the device at the initialization of the tab and can't be exceeded. After pushing the start button the drive executes the predefined movement. The communication with the device is carried out via SDOs.



Profile Position Mode IclA D065

Mode settings

Profile velocity: 741

Profile acceleration: 1325

Profile deceleration: 1576

Target position: 2000

Mode: single set-point

□ abs

Trajectory settings

Target position

2000

Actual position

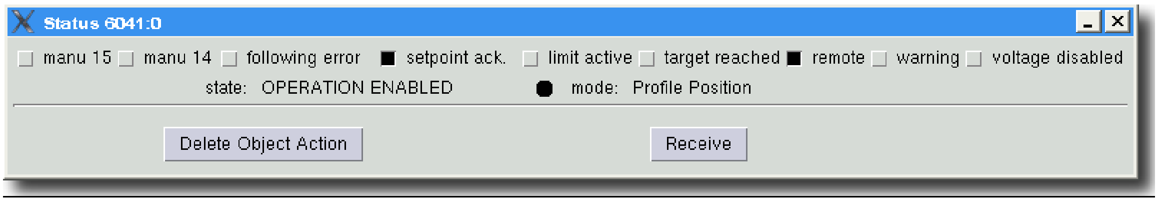
9561

Status: 0x1227

Last Emergency: none

Figure 44: DSP402 Position Mode

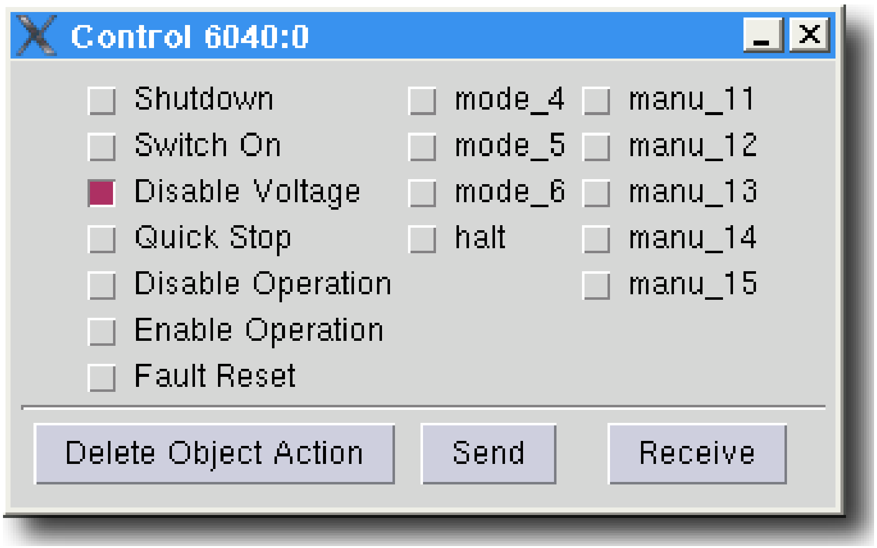
31.4. Object extensions



□ manu 15 □ manu 14 □ following error ■ setpoint ack. □ limit active □ target reached ■ remote □ warning □ voltage disabled

state: OPERATION ENABLED ● mode: Profile Position

Figure 45: Status word bit box



□ Shutdown
□ Switch On
☑ Disable Voltage
□ Quick Stop
□ Disable Operation
□ Enable Operation
□ Fault Reset

□ mode_4
□ mode_5
□ mode_6
□ halt

□ manu_11
□ manu_12
□ manu_13
□ manu_14
□ manu_15

Figure 43: Control word bit box

These extensions are special bit boxes for the status and the control word.

32. About & Release Info Dialog

32.1. About Dialog

The about dialog provides information about:

- the current release
- the type of this release
- the licensee,
- the license and
- the available Tcl/Tk packages.

32.2. Latest Release Info Dialog

When requesting information about the latest release of the CANopen Device Monitor, a http connection to our server is established and the data (approx. 30 bytes) are downloaded from the server.

NOTHING (except your IP address) IS SENT TO THE SERVER WHEN RETRIEVING THE LATEST RELEASE INFO.

33. CDM Command syntax

33.1. SDO commands

`r  <index> <subindex> <typ>`

Description:

SDO read procedure

Parameters:

index	object index
subindex	object subindex
typ	datatype of object <u8|u16|u32|i8|i16|i32|r32|vs>

Results:

read	value

`rr  <index> <subindex> <typ>`

Description:

SDO read procedure: type out the SDO read command and the answer from the device

Parameters:

index	object index
subindex	object subindex
typ	datatype of object

Results:

read	value

`rrc  <index> <subindex> <typ> <ref>`

Description:

SDO read and compare procedure: type out the SDO read command, type out the answer from the device and compare the received value with the reference value

In case of an error the global variable test_error is set to 1.

Parameters:

index	object index
subindex	object subindex
typ	datatype of object
ref	reference value

Results:

0	received value is equal to the reference value
1	received value differs from the reference value

rre <index> <subindex> <typ>

Description:

SDO read and message error: type out the SDO read command, type out the answer from the device and check the received value

Parameters:

index	object index
subindex	object subindex
typ	datatype of object

Results:

0	received no SDO abort domain transfer
1	received SDO abort domain transfer

w <index> <subindex> <typ> <val>

Description:

SDO write procedure

Parameters:

index	object index
subindex	object subindex
typ	datatype of object <u8|u16|u32|i8|i16|i32|r32|vs>
val	value

Results:

OK	value has been written to the object
ERROR*	CiA 309-3 error code if SDO transfer failed

`ww  <index> <subindex> <typ> <val>`

Description:

SDO write procedure: type out the SDO write command and the answer
from the device

Parameters:

index	object index
subindex	object subindex
typ	datatype of object
val	value to write

Results:

nothing

`wwc  <index> <subindex> <typ> <val> <expected>`

Description:

SDO write and compare the expected result
the returned value is compared with the expected one.
typical a write can return "OK" or some errors beginning with "error"

one or two lines with the command and the result are printed to stdout.
the last line contains a right justified flag for
OK - the returned value matches the expected
FAILURE - the returned value does not matche the expected

In case of an error the global variable test_error is set to 1.

Parameters:

index	object index
subindex	object subindex
typ	datatype of object
val	value
expected	the expected return string

Results:

0	received value is equal to the expected value
1	received value differs from the expected value

`wwe  <index> <subindex> <typ> <val>`

Description:

SDO write and message error: type out the SDO write command, type out the answer from the device and check the received value

Parameters:

index	object index
subindex	object subindex
typ	datatype of object
val	value

Results:

0	received no SDO abort code
1	received SDO abort code

33.2. PDO commands

```
::pdo::set_rpdo  <scope> <pdo_nr> <cob> <trans> <index1>
<sub1> <indexN> <subN>
```

Description:

Defines a RPDO at the device and a TPDO at the gateway
Example:
`::pdo::set_rpdo local 1 0x220 event 0x6200 1 0x6200 2`
This defines a PDO to set the first two 8-bit ports on a digital output device according CiA 401. Index and sub-index are the destination objects, which must be available in the EDS file of the current device.
See `::pdo::wpdo` as a usage example.

Parameters:

scope	local|remote (local .. configure only CANopen-Gateway)
pdo_nr	number of pdo
cob	cob id for this pdo
trans	transmission type (event|rtr|sync<1..240>)
index1	1st index to be mapped (format 0x%4X)
sub1	1st sub to be mapped ..
indexN	nth index to be mapped
subN	nth sub to be mapped ..

Results:

-

```
::pdo::set_tpdo  <scope> <pdo_nr> <cob> <trans> <index1>
<sub1> <indexN> <subN>
```

Description:

Defines a TPDO at the device and a RPDO at the gateway
Example:
`::pdo::set_tpdo local 1 0x220 event 0x6000 1 0x6000 2`

Parameters:

scope	local|remote (local .. configure only CANopen-Gateway)
pdo_nr	number of pdo
cob	cob id for this pdo
trans	transmission type (event|rtr|sync<1..240>)
index1	1st index to be mapped (format 0x%4X)
sub1	1st sub to be mapped .. decimal
indexN	nth index to be mapped
subN	nth sub to be mapped ..

Results:

-

`::pdo::wpdo  <num> <length> <args>`

Description:

Transmits a predefined PDO

It must be configure with set_rpdo before!

Example:

```
::pdo::wpdo 1 2 0xff 0xaf
```

Parameters:

num	tpdo number
length	number of data items
args	value1 value2 ... valueN

Results:

-

`::pdo::wwpdo  <num> <length> <args>`

Description:

PDO write procedure: type out the PDO write command

Parameters:

num	PDO number
length	number of data items
args	value1 value2 ... valueN

Results:

nothing

::pdo::rpdo <num>

Description:

Requests a predefined PDO by rtr

It must be configure with set_tpdo before!

Example:

```
::pdo::rpdo 63
```

Parameters:

num	number of PDO

Results:

-

::pdo::setEcatTpdo <num> <node> <nodePdoNum> <mapCnt> <dataTypes>

Description:

Configure EcatServer to forward TPDO of the device to the EDM

Example:

```
::pdo::setEcatRpdo 1 27 2 4 u8 u16 u16 u8
```

Parameters:

num	number of pdo in EcatServer
node	Slave node-Id
nodePdoNum	pdo number at node
mapCnt	number of mapped objects
dataTypes	list of dataTypes which are mapped ..

Results:

Returns	OK or error message

::pdo::setEcatTpdo <num> <node> <nodePdoNum> <mapCnt> <dataTypes>

Description:

Configure EcatServer to send data to a RPDO at the device
Example:

```
::pdo::setEcatRpdo 1 27 2 4 u8 u16 u16 u8
```

Parameters:

num	number of pdo in EcatServer
node	Slave node-Id
nodePdoNum	pdo number at node
mapCnt	number of mapped objects
dataTypes	list of dataTypes which are mapped ..

Results:

Returns	OK or error message

`::pdo::resetEcatTpdo`

Description:

This proc resets all configured TPDOs at the EcatServer

Parameters:

-

Results:

-

`::pdo::resetEcatRpdo`

Description:

This proc resets all configured RPDOs at the EcatServer

Parameters:

-

Results:

-

`::pdo::setPDOCycle  <time_us>`

Description:

The EcatServer as EtherCAT Master exchanges the PDO data with all connected slaves at a given interval which is called here ’PDO cycle’. This PDO cycle can be configured with that function

Parameters:

time_us	time for PDO cycle in microseconds

Results:

nothing

`pdo::setPDOTransmission  <pdoNr> <time_ms>`

Description:

Process data from the slaves are transfered to the EDM not at every PDO cycle but with a configurable interval to reduce the load the EDM. This interval can be configured in milliseconds or a value of 0 indicates an event-driven transmission.

Parameters:

pdoNr	number of PDO in EcatServer configuration
time_ms	time for PDO update in milliseconds

Results:

nothing

`::pdo::setHandler  <num> <cmd>`

Description:

Registers a PDO indication function for PDO

The PDO indication function <cmd> is called when PDO number <num> arrives. The function <cmd> is called with three arguments:
`num` PDO number
`dl` number of data
`data` list of data

Parameters:

num	number of PDO
cmd	name of PDO indication function

Results:

-

`::pdo::setPDOIndication  <num> <cmd>`

Description:

Registers a PDO indication function for PDO

::pdo::setPDOIndication is a compatibility alias to ::pdo::setHandler The PDO indication function <cmd> is called when PDO number <num> arrives. The function <cmd> is called with three arguments:
`num` PDO number
`dl` number of data
`data` list of data

Parameters:

num	number of PDO
cmd	name of PDO indication function

Results:

-

`::pdo::waitForPDO  <pdoNr> <script>`

Description:

Waits for a PDO and evals a script
when the PDO arrives

Parameters:

pdoNr	number of PDO
script	(opt.) optional script that must return 1, if it exists. Within the script the variables val(1),val(2) .. val(n) can be used. They contain the values of mapped objects in the PDO. val(1) refers to the 1st object. val(2) to the 2nd, and so on.

Results:

-

33.3. NMT commands

`::nmt::start <node>`

Description:

Sends the NMT command ’start’ to one or all nodes

Parameters:

node	node-ID (0 addresses the whole network)

Results:

-

`::nmt::stop <node>`

Description:

Sends the NMT command ’stop’ to one or all nodes

Parameters:

node	node-ID (0 addresses the whole network)

Results:

-

`::nmt::preop <node>`

Description:

Sends the NMT command ’enter pre-operational’ to one or all nodes

Parameters:

node	node-ID (0 addresses the whole network)

Results:

-

`::nmt::resetComm <node>`

Description:

Sends the NMT command ’reset communication’ to one or all nodes

Parameters:

node	node-ID (0 addresses the whole network)

Results:

-

`::nmt::resetAppl  <node>`

Description:

Sends the NMT command ’reset node’ to one or all nodes

Parameters:

node	node-ID (0 addresses the whole network)

Results:

-

`::nmt::resetServer`

Description:

It reset the ECAT server and rebuilds PDO data.

Parameters:

-

Results:

-

33.4. LSS commands

`::lss::baud2Index  <baudrate>`

Description:

Converts the baudrate as number (e.g. 125 or 1000) into
the LSS table index

Parameters:

baudrate	baudrate as number

Results:

returns	baud rate index

`::lss::setHandler  <cmd>`

Description:

set callback function to be called when ever a LSS message
UNCONFIGURED DEVICE is received.

Parameters:

cmd	the name of the Tcl procedure

Results:

-

33.5. Other CANopen commands

`::cdm::setSDOTimeOut  <node> <time>`

Description:

Sets the SDO time-out time for a specific node

Parameters:

node	node-ID of the node
time	time out time in ms

Results:

-

`::cdm::enableGuarding  <node> <gtime> <ltime>`

Description:

Starts the guarding of a node

Parameters:

node	node-ID of the node that should be guarded
gtime	guarding time
ltime	life time factor

Results:

-

`::cdm::disableGuarding  <node>`

Description:

Stops the guarding of a node

Parameters:

node	node-ID of the node that had been guarded

Results:

-

`::cdm::setHeartbeat  <time>`

Description:

Configures the heartbeat producer interval of the CANopen-Gateway

Parameters:

time	interval in ms

Results:

-

`::cdm::enableHeartbeat  <node> <time>`

Description:

Configures the heartbeat consumer time for a given node

Parameters:

node	node-Id of monitored node
time	interval in ms

Results:

-

`::cdm::enableSync  <sync>`

Description:

if no argument is provided, it asks the user for a sync intervall and enables sync

Parameters:

sync	sync intervall in ms (opt.)

Results:

-

`::cdm::readU64  <index> <subindex>`

Description:

Read a u64 variable as octet string
and return a string like 0x1234567812345678

Parameters:

index	index
subindex	subindex

Results:

Returns	a u64 value

::cdm::writeU64 <index> <subindex> <value>

Description:

Write a u64 variable as octet string

Parameters:

index	index
subindex	subindex
value	u64 value

Results:

-

::cdm::status <fileName>

Description:

Print out a lot of internal status informations
It might be useful for support purpose

Parameters:

fileName	file name for output (opt.)

Results:

-

::cdm::disableSync

Description:

Disables the transmission of SYNC messages

Parameters:

-

Results:

-

`::emcy::setHandler  <cmd>`

Description:

Registers a indication function for emergencies

The indication function is called when an Emergency message arrives.
It has these arguments:
node - node id
args - emergency

Parameters:

cmd	name of indication function

Results:

-

`::sdo::setHandler  <cmd>`

Description:

Registers a indication function for sdo write indications

The indication function is called when a device writes to the object dictionary of the CANopen server.
It has these arguments:
index - index
sub - subIndex
len - length of data (or b64 data)
data - data
For nun-numerical data base64-encoding is used.

Parameters:

cmd	name of indication function

Results:

-

`::dcf::downloadDCF  <edsFile> <dcfFile> <nodeId>`

Description:

This proc loads a DCF file and downloads its data to a given node

Parameters:

edsFile	path to EDS File
dcfFile	path to DCF File
nodeId	node id of device

Results:

-

`::cdm::pause  <msecs>`

Description:

Waits for a number of milliseconds with event handling

Parameters:

msecs	ms to wait

Results:

-

`::hbt::setHandler  <cmd>`

Description:

Registers a indication function for heartbeat events

The indication function is called when a heartbeat
event like BOOT-UP, HBT started or HBT lost arrives.
It has these arguments:
node - node id
data - data from 309-3 server

Parameters:

cmd	name of indication function

Results:

-

`::hbt::unsetHandler`

Description:

Unregisters the indication function for heartbeat events

Parameters:

-

Results:

-

33.6. Test commands

`::cdm::addTab  <titel> <pos>`

Description:

adds a tab into the Tabset of the right side

Parameters:

titel	name of the Tab displayed at the ...
pos	position starting with 0, can be end

Results:

frame	the window name of the top level frame within this tab

`::cdm::deleteTab  <pos>`

Description:

deletes a tab from the tabset

Parameters:

pos	position starting with 0, can be end

Results:

`::cdm::addTestTabOld  <filename> <title>`

Description:

adds a special tab on the right side of the device monitor;
the tab-card contains checkbuttons for test-scripts,
the list with scripts is in a file and will set by user

Parameters:

filename	name of the file which contains a list of scripts for the special tab
title	title of the tab

Results:

window	path to Tab

::cdm::stringCenter <string> <l>

Description:

center string -- prepend spaces to a given string
if the result will be printed it looks like it is centered within
a line lenght of l

Parameters:

string	unformated string
l	desired line length

Results:

centered	string

::cdm::stringFill <string> <endword> <l>

Description:

append spaces and endword at string until line length l

Parameters:

string	unformated string
endword	optional END-word, defaults to {}
l	desired line length, defaults to 80

Results:

formatted	string

::cdm::banner

Description:

prints a headline with device and user characteristic

Parameters:

nothing

Results:

nothing

::cdm::putsDateTime

Description:

prints the current date and time

Parameters:

nothing

Results:

nothing

::cdm::commentInput <wtitle> <cancelstring>

Description:

user input for comments

Parameters:

wtitle	window title
cancelstring	cancel string

Results:

nothing

::cdm::userDialog <title> <type>

Description:

user response dialog
if a dialaog is finished with "Not Ok" another dialogbox
for giving a reason is opened

Parameters:

title	additional text for displaying to the user
type	specifies the type of dialogue
-	0 ... only wait for OK
-	1 ... decide between OK and NotOk
-	2 ... decide between OK NotOk and Abort
-	3 ... decide between OK and NotOk without comment
-	4 ....decide between list of given choices.

Results:

0	decided for OK
1	decided for Not OK
2	decided for Abort
or	selected button(text) for type 4

33.7. CDM commands

`::cdm::getObjectType  <node> <index>`

Description:

Returns the objectType

Parameters:

node	node-ID ({} means current node ID) as %d or 0x%x
index	index as %X, or 0x%x

Results:

Returns	objectType (VARIABLE,ARRAY,RECORD,DOMAIN)
throws	an error, if the index does not exist

`::cdm::getDataType  <node> <index> <sub>`

Description:

Returns the data type of a sub index

Parameters:

node	node-ID ({} means current node ID) as %d or 0x%x
index	index as %X, or 0x%x
sub	subindex (opt) as %03d, %0x or 0x%x

Results:

Returns	DataType (u8|u16|u32|i8|i16|u32|r32|vs)
throws	an error, if the index does not exist

`::cdm::getName  <node> <index> <sub>`

Description:

Returns the parameter name of an object from the EDS

Parameters:

node	node-ID
index	index
sub	sub index (opt) as %03d, %0x or 0x%x

Results:

returns	the parameter name or throws an error if the object does not exist

`::cdm::getDefaultValue  <node> <index> <sub>`

Description:

Returns the default value of an object from the EDS

Parameters:

node	node-ID
index	index
sub	sub index (opt) as %03d, %0x or 0x%x

Results:

returns	the default value, throws an error if the object does not exist

`::cdm::existObject  <node> <index> <sub>`

Description:

Checks, if an object exists in the EDS file

Parameters:

node	node-ID ({} means current node ID) as %d or 0x%x
index	index as %X, or 0x%x
sub	subindex (opt) as %d, 0x%x

Results:

1	object exists
0	object does not exist

`::cdm::getRemoteID`

Description:

Returns the current remote ID

Parameters:

-

Results:

Returns	the remote-id

`::cdm::setRemoteID <id>`

Description:

Sets the remote id and updates the OD tree

Parameters:

id	remote node id

Results:

1	success
0	invalid ID

`::cdm::loadEds <fileName>`

Description:

Loads an EDS file for a node and after that
it looks for a description file with a matching name
and loads it.
If there is a matching device-specific .rc-file
it is sourced to. This file must contain valid Tcl or CDM commands

Parameters:

fileName	path to EDS file (POSIX style)

Results:

-

`setMessageLogLimit <limit>`

Description:

Set a new limit of lines for the message log

Parameters:

limit	max number of lines in message log

Results:

Returns	OK if limit is ok, otherwise a verbose error

`::cdm::hideGUI  <flag>`

Description:

Hides the CDM GUI.
This function is useful for scripts that build their own user interface.

Parameters:

flag	(opt.) use -noconsole to exclude the console window

Results:

-

`::cdm::showGUI  <flag>`

Description:

Shows the CDM GUI. Counterpart of ::cdm::hideGUI
This function is useful for scripts that build their own user interface.

Parameters:

flag	(opt.) use -noconsole to exclude the console window

Results:

-

`isColor  <color>`

Description:

Checks if a color is a valid tcl color

Parameters:

color	color name or #hex expression

Results:

1	is valid color
0	is no valid color

`int2asc  <i>`

Description:

Converts an unsinged char value into a ASCII representation

Parameters:

i	unsigned char value

Results:

Returns	an ascii value

`int2bits  <i> <digits>`

Description:

Converts an integer value into a binary representation like 0b01010101

Parameters:

i	integer value
digits	length of the returned value (opt.)

Results:

Returns	a binary value

`::common::every  <script> <ms>`

Description:

This proc runs a script cyclically.
The global variable every(script) stores the after-id
for each script.

Parameters:

script	script to run
ms	interval in ms (opt.) defaults to 1000

Results:

-

`clear_messagelog`

Description:

This proc deletes the content of the message log

Parameters:

-

Results:

-

`save_messagelog <filePath>`

Description:

This proc saves the content of the message log
into a file

Parameters:

filePath	path to writable file

Results:

-

`clear`

Description:

This proc deletes the content of the CDM console

Parameters:

-

Results:

-

`tkcon save  <filePath> <spec>`

Description:

This command saves the content of the CDM console
to a file

Parameters:

filePath	path to writeable file
spec	content specifier (use all for all content)

Results:

-

33.8. DSP402 commands (DSP402-Extension)

`::402::ret`

Description:

List of possible return values

Parameters:

-

Results:

0	OK
1	Drive in wrong state
2	Transition not possible
3	SDO abort occured
4	No setpoint acknowledge

`::p402::ppHandleNewSetpoint  <delayTime>`

Description:

This function handles a new setpoint in the Profile Position mode.
The function initiates an absolute movement
in the single-setpoint mode.
If this function returns with an error
it is possible that the mode-specific bits in
object 0x6041 (controlword) and object 0x6041
(statusword) are not cleared.
Each drive needs a specific time for the transfer
of a new setpoint. This time can be specified by
the argument delayTime.
The communication is done via SDO.
This function can only be used for single drive devices.
This function is used by other functions of this
namespace.

Parameters:

delayTime	time for transfer in ms

Results:

ret	see variable ::p402::ret

::p402::getState

Description:

This function gets the actual CiA-402 state of the drive.
The actual CiA-402 state is returned
in the format of object 0x6041 (statusword).
The communication is done via SDO.
This function can only be used for single drive devices.

This function returns a list with the following elements: retsee variable ::p402::ret stateactual CiA-402 state in statusword format The state is only valid if this function returns with ok.

Example for usage:

```
set retList [::p402::getState]
set retVal [lindex \$retList 0]
if { [set retVal] != [set ::p402::ret(OK)] } {
 puts "Error: getState() returns with [set retVal]."
}
set actualState [lindex \$retList 1]
```

If actualState is 0x0027 the drive is in the state OPERATION ENABLED.

Parameters:

-

Results:

retList	value of ::p402::ret + actual CiA-402 state

::p402::changeState <state> <delayTime>

Description:

This function changes into the desired CiA-402 state.
The desired CiA-402 state must be input
in the format of object 0x6041 (statusword).
Each drive needs a specific time to change the CiA-402 state.
This time is set by the argument delayTime.
The communication is done via SDO.
This function can only be used for single drive devices.

Example: The argument state must be 0x0027 for
a change into the CiA-402 state OPERATION ENABLED.

Parameters:

state	desired CiA-402 state
delayTime	maximal time for state changing in ms

Results:

ret	see variable ::p402::ret

`::p402::halt`

Description:

This function activates the halt function,
i.e. the motion is halted.
The Halt bit in object 0x6040 (controlword) is set.
To reset Halt use the function ::p402::<mode>Change,
because the necessary functionality to reset Halt is
mode-specific.
The communication is done via SDO.
This function can only be used for single drive devices.

Example for reset Halt:
use ::p402::ppChange for the pp mode

Parameters:

-

Results:

ret	see variable ::p402::ret

`::p402::modeStop  <delayTime>`

Description:

This function stops a motion by the CiA-402 state transition
from the CiA-402 state OPERATION ENABLED into the CiA-402
state SWITCHED ON. The drive is stopped.
The communication is done via SDO.
This function can only be used for single drive devices.

Note: The operation mode is not changed to NO_MODE,
because not all drives support NO_MODE.

Parameters:

delayTime	maximal time for state changing in ms

Results:

ret	see variable ::p402::ret

`::p402::pvStart <targetVelocity> <profileAcceleration> <delayTime>`

Description:

This function starts a motion in the Profile Velocity mode.
The Profile Velocity mode is configured by the
mandatory objects of the pv-mode.
The operation mode is set to Profile Velocity.
The motion is started by the change into the CiA-402 state
OPERATION ENABLED.
The communication is done via SDO.
This function can only be used for single drive devices.

Parameters:

targetVelocity	value of object 0x60FF
profileAcceleration	value of object 0x6083
delayTime	maximal time for state changing in ms

Results:

ret	see variable ::p402::ret

`::p402::pvChange <targetVelocity>`

Description:

This function changes the velocity of the movement in the
Profile Velocity mode.
The communication is done via SDO.
This function can only be used for single drive devices.

Parameters:

targetVelocity	value of object 0x60FF

Results:

ret	see variable ::p402::ret

`::p402::ppStart <targetPos> <profileVel> <profileAcc> <delayTime>`

Description:

This function starts a motion in the Profile Position mode.
The Profile Position mode is configured by the mandatory objects of the pp-mode.
The operation mode is set to Profile Position.
This function initiates an absolute movement in the single-setpoint mode.
The motion is started by the change into the CiA-402 state OPERATION ENABLED and the execution of the new-setpoint handling.
Each drive needs a specific time for the transfer of a new setpoint. This time can be specified by the argument delayTime.
The communication is done via SDO.
This function can only be used for single drive devices.

Parameters:

targetPos	value of object 0x607A
profileVel	value of object 0x6081
profileAcc	value of object 0x6083
delayTime	transfer time in ms

Results:

ret	see variable ::p402::ret

`::p402::ppChange <targetPosition> <delayTime>`

Description:

This function changes the target position in the Profile Position mode.
This function initiates an absolute movement in the single-setpoint mode.
The movement is started by the execution of the new-setpoint handling.
Each drive needs a specific time for the transfer of a new setpoint. This time can be specified by the argument delayTime.
The communication is done via SDO.
This function can only be used for single drive devices.

Parameters:

targetPosition	value of object 0x607A
delayTime	transfer time in ms

Results:

ret	see variable ::p402::ret

34. Appendices

Appendix 1 — CANopen Commands in Overview

Node Guarding

- ::cdm::enableGuarding <node> <gtime> <ltime>
- ::cdm::disableGuarding <node>

PDO

- ::pdo::set_rpdo <scope> <pdo_nr> <cob_id> <trans> <map_index1> <map_sub1> ...
  <map_indexn> <map_subn>
- ::pdo::set_tpdo <scope> <pdo_nr> <cob_id> <trans> <map_index1> <map_sub1> ...
  <map_indexn> <map_subn>
- ::pdo::wpdo <pdo_num> <length> <args>
- ::pdo::rpdo <pdo_num>
- ::pdo::waitForPdo <pdo_num> <script>
- ::pdo::setHandler <pdo_num> <proc>
- ::pdo::setPDOIndication <pdo_num> <proc> (compatibility alias to setHandler)

Reset Application

- ::nmt::resetAppl <node>

Reset Communication

- ::nmt::resetComm <node>

SDO

- r <index> <subindex> <type>
- rr <index> <subindex> <type>
- rrc <index> <subindex> <type> <ref_val>
- rre <index> <subindex> <type>
- w <index> <subindex> <type> <val>
- ww <index> <subindex> <type> <val>
- wwc <index> <subindex> <type> <val> <ref_val>
- wwe <index> <subindex> <type> <val>

SYNC

- ::cdm::enableSync <cycle>
- ::cdm::disableSync

Communication state change

- ::nmt::preop <node_id>

- ::nmt::start <node_id>
- ::nmt::stop <node_id>

Node Monitoring

- ::hbt::setHandler <callback>
- ::hbt::unsetHandler

Appendix 2 — CDM Commands in Overview

Date/Time

- ::cdm::putsDateTime

Dialog

- ::cdm::userDialog <title> <type>

Download

- ::cdm::domainDownload <node> <index> <sub> <timeout in ms> <file>
- ::cdm::domainUpload   <node> <index> <sub> <timeout in ms> <file>

Network

- ::cdm::profileScan <dprf> <timeout>
- ::cdm::getRemoteID
- ::cdm::setRemoteID <node>

EDS

- ::cdm::loadEDS <fileName>

Object dictionary access

- ::cdm::getDataType {<node>} <index> {<sub>}
- ::cdm::getObjectType {<node>} <index>
- ::cdm::getDefaultValue {<node>} <index> {<sub>}

Test Protocol Header

- ::cdm::banner

Tabbed Fields

- ::cdm::addTab {<tab_text> {<pos>}}
- ::cdm::addTestTab t_<conf> {<tab_text>}
- ::cdm::deleteTab {<pos>}

Formatting of Strings

- ::cdm::stringCenter <string> <line length>
- ::cdm::stringFill <string> <endword> <line length>

Appendix 3 — Creation of an Object Description

An object description ("<working directory>\<device>.txt") has an entry for every object. This entry consists of:

1.) the index (4-digit, hexadecimal) followed by ’:’

2.) the EDS name

3.) an empty line and

4.) the description

The description is not limited in the length. The text of the description does not allow to contain an index (see 1.) at line start.

```
1A00:
Transmit PDO Mapping Parameter

It contains the mapping parameter for the PDOs the
device is able to transmit.
Sub-index 0 contains the number of the mapped
data objects. All further entries define the data by
it’s index, sub-index and length.
The structure of a mapping entry is:

index,subindex,length

1A14:
Transmit PDO Mapping Parameter
 ......
```

Literature

[1] Tcl and the Tk Toolkit
Ousterhout, John K.
Addison-Wesley, 1994
ISBN 0-201-63337-X

[2] Practical Programming in Tcl and Tk, 2d ed.
Welch, Brent
Prentice Hall, 1997

[3] Tcl/Tk Tools
Harrison, Mark
O'Reilly & Associates, 1997

[4] Effective Tcl/Tk Programming
Harrison, Mark; McLennan, Michael
Addison-Wesley, 1998

Literature in the Internet

[5] http://www.tcl.tk

[6] http://wiki.tcl.tk

[7] http://www.activestate.com/solutions/tcl/

[8] http://incrtcl.sourceforge.net/blt/index.html
The BLT Toolkit.
BLT is an extension to the Tk toolkit, adding new widgets, geometry managers, and miscellaneous commands.

[9] Tcl and the Tk Toolkit
Ousterhout, John K.; Jones, Ken
Addison-Wesley, 2010, second edition
ISBN 978-0-321-33633-0

CANopen Device Monitor Wiki

[11] http://www.can-wiki.info/CanOpenDeviceMonitor
The CANopen Device Monitor Wiki is a constantly growing collection of sample scripts and tips.

35. Glossary

CAN	Controller Area Network
CAN FD	Controller Area Network - Flexible Data Rate
CAL	CAN Application Layer (CANopen base)
CDM	CANopen Device Monitor
CiA	CAN in Automation international users and manufacturers group e.V.
CN	Controlled Node (Ethernet POWERLINK)
COB	Communication Object (CAN Message)
COB-ID	Communication Object Identifier
CSDO	Client SDO
EDM	EtherCAT Device Monitor
EDS	Electronic Data Sheet
ESI	EtherCAT Slave Information file
EMCY	Emergency Object
EPSG	Ethernet POWERLINK Standardization Group
ETG	EtherCAT Technology Group
MN	Managing Node (Ethernet POWERLINK)
NMT	Network Management
OD	Object Dictionary
PDO	Process Data Object, unconfirmed service for real time communication
RPDO	Receive PDO
RTR	Remote Transmission Request
PDM	POWERLINK Device Monitor
PRMS	Problem Report Management System
SDO	Service Data Object, Confirmed data transfer service for parameter data.
SSDO	Server SDO
SYNC	Sychronization Object
Tcl	Tool Command Language (script language)
TCP/IP	Transmission Control Protocol/Internet Protocol
TIME	Time Stamp Object
Tk	Tcl Tool kit (graphical Tcl extension), Tcl/Tk
TPDO	Transmit PDO
Widget	element of a graphical user interface (e.g. button, entry filed, menu, ...)

36. Index

- a -

Action
    tab- 43
    tab-set 20
activate_bitrate, LSS 74
after 97

- b -

base64 47
bitwise representation 45

- c -

CAN bit rate 15
cdm::addTab 52
cdm::addTestTab 49
cdm::deleteTab 53
cdm::domainDownload 66
cdm.rc 15
commands, Tcl 58

- d -

DCF-file 54
description, object 118
device change 15
device.rc 16
device settings 16
doc-file 118

- e -

editor 15
EDS file 15

- f -

file, project 69

- g -

get_node, LSS 73
global_stop, variable 51

- i -

identity, LSS 73
index 20

- l -

LoadConfig 57
logging 63
LSS
    activate_bitrate 74
    get_node 73
    identity 73
    set_bitrate 73
    set_node 73
    store 74
    switch_glob 73
    switch_sel 73

- m -

menu online 56
Message Log 63

- n -

network 68
NMT tab 31
NOTEPAD 15

- o -

object
    configuration file 57
    description 33, 118
OCF-file 57
octet 46
offline 56
online 56
Options 28

- p -

pause 97
PDO 36, 60
preop 31
Preop All 31
Preop Node 31
procedure, Tcl 59
profile position mode 77
program download 66
    SDO 66
project file 69

- r -

Repository 15

- s -

SaveConfig 57
Scan Network 31
scripting, Tcl 64
script load 59
SDO program download 66
Send Sync 31
set_bitrate, LSS 73
set_node, LSS 73
slider 43–44
start 31
Start All 31
Start Node 31
stop 31
store, LSS 74
strip chart 41
subindex 20
switch_glob, LSS 73
switch_sel, LSS 73
system requirements 8

- t -

t_start.tcl 48
tab-, Action 43
tab, NMT 31
tab-set, Action 20

tabs, user-specific 48
Tcl
    commands 58
    procedure 59
    scripting 64
tree 20

- u -

update 51
user-specific tabs 48

- v -

variable global_stop 51

- w -

wait 97

Table of Contents

1. Introduction . . . 7

1.1. Product Overview . . . 7
1.2. Conventions . . . 8

2. Installation . . . 10

3. Quick Start . . . 11

3.1. Preparations . . . 11
3.2. First program start . . . 11
3.3. Communication with a CANopen device . . . 13

4. Configuration . . . 15

4.1. Setting of the CAN Bitrate . . . 15
4.2. Loading of the Object Dictionary . . . 15
4.3. Device Specific Settings . . . 16
4.4. Start-Up Scripts . . . 16

5. CAN-Interface Configuration . . . 18

5.1. Configuration Dialog . . . 18

6. Object Dictionary Accesses . . . 20

7. Menu Structure . . . 23

7.1. File . . . 23
7.2. Edit . . . 23
7.3. View . . . 24
7.4. Connection . . . 24
7.5. Extras . . . 25
7.6. Windows . . . 26
7.7. Help . . . 26

8. Toolbar . . . 27

9. Options . . . 28