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
© 0 port GmbH
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
1.

1.1.
The EtherCAT® Device Monitor (EDM) communicates with EtherCAT® devices in EtherCAT® networks by using of various EtherCAT® services. Therewith the Device Monitor supports:

- development,
- diagnostic,
- implementation and configuration

of EtherCAT® devices.

The EtherCAT® Device Monitor uses an EtherCAT® server to communicate with the EtherCAT® devices. The communication between the graphical user interface and the server is done by the TCP/IP protocol.

The EtherCAT server is as an EtherCAT Master and it can run on the local PC as the EtherCAT® Device Monitor or on a separate device.

**Figure 1**: Structure of the EtherCAT® Device Monitor

The EtherCAT® 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,
- no device configuration by DCF file,
- support of only some fixed node IDs
• no support for FoE and EEPROM access
• run-time limited to 30 minutes

The EtherCAT® 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

1.2.
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’.
<k> 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.

1.3.

port supports the user by telephone hot-line and by seminars. Additionally port offers consultations in the whole field of EtherCAT® e.g. network planning, network configuration, selection of devices and EtherCAT® and CANopen Profile implementations.
2.1. The most common use case of the EtherCAT® Device Monitor is the is that the EtherCATServer is used locally on the same PC as the EtherCAT® Device Monitor. Prior to the first usage of the tool the must be configured to start the EtherCATServer automatically. To do so, open the interface configuration dialog at Connection -> EtherCAT®-Interface. Specify "Local EtherCATServer" in the field "EtherCAT-Interface and below the path to the file ecatserver.exe has to be entered. Finally, the physical Ethernet interface device has to be specified, which shall be used by the EtherCAT® Device Monitor to access the EtherCAT network.

For the second possible use case, using an EtherCATServer whichs runs on a remote PC or embedded device, "TCP" has to be selected as EtherCAT interface and its IP address or host name must be specified at the input field named "host"

For both use cases, ensure that the EtherCATServer is connected with the EtherCAT network with a cable connection and that the right ethernet interface is specified, if the PC or embedded device supports more than one interface.

Figure 7, EtherCAT® Interface Configuration
After the configuration the connection can be established. Click on the first symbol in the toolbar to connect to the device.

Figure 8, Connect button in tool bar
After the connection is established, the EtherCAT® gateway validates the license on the gateway. If the license is not valid, the EtherCAT® Device Monitor can only communicate with a few nodes (demo functionality).
The EtherCAT® Device Monitor connects to the EtherCAT® server now and at success the online status is displayed in the status bar and the background of the entry field turns pink.

![Image](image1.png)

**Figure 3:** EtherCAT® Device Monitor

! A valid configuration of the hardware interface is saved by the EtherCAT® 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 EtherCAT® Device Monitor connects automatically at every start of the program.

2.2.

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

![Image](image2.png)

**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 EtherCAT® Device Monitor calculates them
and transmits the result to the device. However, the expression may not contain any blanks.

Figure 5: Formulas in the entry
The "NMT tab" serves for sending NMT commands. This tab can be torn off like every tab from the anchorage in the EtherCAT® Device Monitor out into a separate window. To tear it off, click on the dashed line above the tab.

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.
3.

The object dictionary is the data interface between the EtherCAT® Device Monitor and the EtherCAT® device. The EtherCAT® Device Monitor can access every object in the object dictionary for reading or writing by index/subindex.

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

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

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

**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 EtherCAT® 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.
4.

4.1.

Load EDS
Load default EDS
Recent EDS files
Load device configuration
Save device configuration
Project
Exit

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 EtherCAT® Device Monitor

4.2.

Cut
Copy
Paste

Cut
Cuts selected text into clipboard
Copy
Copies selected text into clipboard
Paste
Pastes text from clipboard
4.3.

*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)

4.4.

*Connect*  
Connects to the EtherCAT® server

*Disconnect*  
Closes the connection to the EtherCAT® 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 EtherCAT® interface

4.5.
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.

Set DCF component flag for all objects  
Unset DCF component flag for all objects

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 EtherCAT® Device Monitor, but it is no complete EDS file according to the standard.
4.6.

<table>
<thead>
<tr>
<th>Menu Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear All</td>
<td>Clears the console and the log window except of</td>
</tr>
<tr>
<td>Cascade</td>
<td>Cascades all open windows</td>
</tr>
<tr>
<td>Tile vertical</td>
<td>Tiles all open windows vertically</td>
</tr>
<tr>
<td>Tile horizontal</td>
<td>Tiles all open windows horizontally</td>
</tr>
</tbody>
</table>

4.7.

<table>
<thead>
<tr>
<th>Menu Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Help</td>
<td>Shows the on-line help</td>
</tr>
<tr>
<td>About</td>
<td>Shows version and licence information</td>
</tr>
<tr>
<td>Latest Release Info</td>
<td>Fetch information about updates from the internet</td>
</tr>
</tbody>
</table>
5.

The toolbar is explained in the following illustration.

![Toolbar Image]

**Figure 16: Toolbar**

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Connect</strong></td>
<td>Connect to EtherCAT network (EtherCAT server)</td>
</tr>
<tr>
<td><strong>Disconnect</strong></td>
<td>Disconnect connection to EtherCAT network</td>
</tr>
<tr>
<td><strong>Position</strong></td>
<td>Selection of the active node based on its position</td>
</tr>
<tr>
<td><strong>1 ... 2 ...</strong></td>
<td>Shortcut to the last active nodes</td>
</tr>
<tr>
<td><strong>Start</strong></td>
<td>Try to set all slaves into OPERATIONAL</td>
</tr>
<tr>
<td><strong>SafeOp</strong></td>
<td>Try to set all slaves into SAFE-OPERATIONAL</td>
</tr>
<tr>
<td><strong>PreOp</strong></td>
<td>Try to set all slaves into PRE-OPERATIONAL</td>
</tr>
<tr>
<td><strong>Refresh</strong></td>
<td>Re-initialization of the EtherCAT server and update of the PDO configuration of the slaves. If the PDO mapping or assignment has been changed these data have to be read again by the EtherCAT server. This might also be necessary during the development of EtherCAT slaves. The EtherCAT server does not recognize if a CPU is stopped in the debugger but the ESC continues to work. In this case the EtherCAT server has to be re-initialized.</td>
</tr>
<tr>
<td><strong>Abort</strong></td>
<td>Stop processes that take a long time, see also <code>global_stop</code></td>
</tr>
<tr>
<td><strong>Query</strong></td>
<td>Fetch information about updates from the internet</td>
</tr>
<tr>
<td><strong>EtherCAT</strong></td>
<td>Opens the EtherCAT page on port’s website</td>
</tr>
</tbody>
</table>
6.

6.1.

### General Settings

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read object on selection</td>
<td>Read object immediately on selection</td>
</tr>
<tr>
<td>Auto-Connect</td>
<td>Connect to server at start-up</td>
</tr>
<tr>
<td>&lt;Del&gt; deletes complete entry field</td>
<td>&lt;Del&gt; key deletes complete input field in value entry frame.</td>
</tr>
<tr>
<td>Reload last project file at start up.</td>
<td>The lastly used project file is opened automatically at start up of the EtherCAT® Device Monitor.</td>
</tr>
<tr>
<td>Save all settings at exit</td>
<td>All settings (options and connection settings) are saved automatically.</td>
</tr>
<tr>
<td>Reuse last EDS for empty nodes</td>
<td>The last EDS file is used for other node-IDs without assigned EDS file.</td>
</tr>
<tr>
<td>Copy EDS files to project directory</td>
<td>The EDS files are copied from the repository to the project folder, if it is saved.</td>
</tr>
<tr>
<td>EDS repository</td>
<td>Configuration of the directory for the EDS repository</td>
</tr>
</tbody>
</table>
6.2.

<table>
<thead>
<tr>
<th>Set DCF flag on change</th>
<th>sets the DCF component flag at change of an object</th>
</tr>
</thead>
<tbody>
<tr>
<td>Download configuration after DCF import</td>
<td>Automatic download of the configuration after DCF import</td>
</tr>
<tr>
<td>Save configuration after download</td>
<td>Automatic saving of the configuration after download</td>
</tr>
</tbody>
</table>

6.3.
<table>
<thead>
<tr>
<th>SDO Timeout (ms)</th>
<th>SDO timeout in ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>EtherCAT Cycle Time (ms)</td>
<td>Cycle time of the EtherCAT network in ms</td>
</tr>
<tr>
<td>Allow old-style OD Scan</td>
<td>If scanning of the object dictionary using the service 'SDO Info' is not supported, the tool can also try to read each object by object.</td>
</tr>
<tr>
<td>Auto re-scan of network</td>
<td>Rescan of network automatically if topology has changed</td>
</tr>
</tbody>
</table>

6.4.

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

6.5.

This dialogue allows the configuration of specific fonts for different GUI elements.
7.

7.1.

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

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start Network</td>
<td>Set all nodes in the state OPERATIONAL.</td>
</tr>
<tr>
<td>Start Node</td>
<td>Set the node with the active node-id in the state OPERATIONAL.</td>
</tr>
<tr>
<td>Preop Network</td>
<td>Set all nodes in the state PRE-OPERATIONAL.</td>
</tr>
<tr>
<td>Preop Node</td>
<td>Set the node with the active node-id in the state PRE-OPERATIONAL.</td>
</tr>
<tr>
<td>Stop Network</td>
<td>Set all nodes in the state STOPPED.</td>
</tr>
<tr>
<td>Stop Node</td>
<td>Set the node with the active node-id in the state STOPPED.</td>
</tr>
<tr>
<td>Safe Op</td>
<td>Set the node into the state SAFE OP.</td>
</tr>
<tr>
<td>Init</td>
<td>Send the NMT command Init Node.</td>
</tr>
</tbody>
</table>

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

7.2.

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 EDM-commands are available.

This function is only available in the full version. Using the eval version only integrated demo scripts can be loaded.
7.3.  
After scanning the network the buttons for the nodes get different colors.

<table>
<thead>
<tr>
<th>Color</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>background color</td>
<td>no node found</td>
</tr>
<tr>
<td>blue</td>
<td>node found</td>
</tr>
<tr>
<td>yellow</td>
<td>active node during network scan</td>
</tr>
</tbody>
</table>

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.

7.4.  

Figure 17: Device information  
When moving the mouse over found nodes device information are displayed as a tool tip.
8.

8.1.

**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.

8.2.

Because of the restrictions of the old CANopen-based EDS format, 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:

```plaintext
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 EtherCAT® 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 EtherCAT® DesignTool by port creates object description files automatically.
9. With the object overview tab all sub indices of an array or record an be read or written at once.

Figure 19: Index overview
Some restrictions exist when reading or writing sub indices. Sub indices with the data type "domain" cannot be read or written and PDO and PDO mapping 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.
10.
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.

![PDO Tab](image)

**Figure 20: PDO Tab**

After a modification of the PDO mapping or PDO assignment, the network PDO configuration has to be updated by clicking at the icon "Reload Network Configuration" in the toolbar. This is necessary to use the modified PDOs.
The following table describes the buttons in the lower part of the mask.

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

10.1.
After pressing the button "Show in Chart" the following dialog window is opened.

![Chart configuration mask](image)

**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.
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*.

![Device Monitor - Node 64 - Icla DO65: Process](image)

**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 EtherCAT® Device Monitor project via "File → Project → Save Project". When the project is opened again, the process image settings are restored.
12.
The PDO Process Image tab displays the data of TPDOs from the EtherCAT® network. There are 2 ways to add Transmit PDOs of the EtherCAT® 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 EtherCAT® server.

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.

![PDO Process Image](image.png)

**Figure 24:** PDO Process image

The settings of the process image can be saved with the complete EtherCAT® Device Monitor project via File → Project → Save Project. When the project is opened again, the process image settings are restored.
13.
Using the EtherCAT service "File over EtherCAT (FoE)" files of an arbitrary size can be transferred from or to EtherCAT devices. The FoE tab is only active, if the selected device supports the FoE service. Otherwise it is disabled.

![Figure 26: FoE Tab](image)

The FoE tab consists of 2 parts. The upper part controls reading files from the device and the lower part controls writing files to the device. For both services the same set of parameters is required: A file name for the local file on the PC, a file name on the device and a password to enable the FoE access on the device. The password must be a 32-bit hexadecimal value as e.g. 0123ABCD.

The file transfer may take a noticeable time especially when transferring large files. The process is indicated using a progress bar and the EtherCAT® Device Monitor is locked during this time.

To use FoE services from scripts the functions ::foe::readFile <deviceFileName> <password> <localFileName> and ::foe::writeFile <deviceFileName> <password> <localFileName> can be used.
The EtherCAT® Device Monitor supports reading and writing of registers of the EtherCAT Slave Controller (ESC). By pressing the "Read ESC" button, 512 bytes from the ESC are read starting at the specified start address.

### Figure 27: ESC Tab
A register of the ESC can be interpreted by highlighting a byte in the raw register view. Thereafter the interpretation of this register is shown in the field at the bottom of the tab.

To use the ESC services from scripts the functions `::esc::regRead <addr>` and `::esc::regWrite <addr> <value>` can be used.
15.
The EtherCAT® Device Monitor supports reading and writing of data of the EEPROM. By pressing the "Read EEPROM" button 512 words from the EEPROM are read starting at the specified start address.

![Figure 28: EEPROM Tab](image)

To use the EEPROM services from scripts the functions ::eeprom::siiRead <addr> and ::eeprom::siWrite <addr> <value> can be used.
16.

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 `::edm::addChartData`. The procedure is defined as follows:

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

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

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

![Figure 29: strip-chart after two calls to `::edm::addChartData`](image)

**Figure 29**: strip-chart after two calls to `::edm::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]
    edm::addChartData [list $value 0 0 0]
}
%
% # Call this procedure every second
% ::common::every "readAndShow 0x6401 0x01 i16" 1000
%
```

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

The console or script command ::edm::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:

<table>
<thead>
<tr>
<th>Menu entry</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear Chart</td>
<td>Deletes all values from the chart.</td>
</tr>
<tr>
<td>Save Chart</td>
<td>Saves the current picture as PostScript file.</td>
</tr>
<tr>
<td>Save Data</td>
<td>Saves all values as CSV file.</td>
</tr>
<tr>
<td>Stop Updating</td>
<td>Stops the update of the chart</td>
</tr>
</tbody>
</table>
17.

17.1.

Optionally the tabbed field Action can be extended by further elements. A possible element is a slider (Figure 30).

![Figure 30](image)

**Figure 30.** 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. 
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.

![Slider in a top level window](image)

*Figure 31: 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.
Each numerical object can be displayed bitwise in a unique window.

Figure 32: Bit box in a top level window
All object extension can be activated by pressing the right mouse button in the object tree.

Figure 33: Configuration of the GUI Extension
The assignment of slider and bit boxes to particular objects can be saved in an OCF file.
18.

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

There are 2 kinds of tabs available:

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

18.1.

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 EtherCAT® Device Monitor. It provides lists of check-boxes for the user defined test scripts.

![User Defined Tabbed Field with Test Scripts](image)

**Figure 29**: 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 `edm::addTestTab`. Figure 29 shows an example for an additional tab.
edm::addTestTab t_conf {<tab_name>}

Description
Creates an additional tab on the right side of the EtherCAT® 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 29 shows the result.
t_start.tcl:

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

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 EtherCAT® 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

```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 == "" } {
        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 EtherCAT® 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 preparations. 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. Therefore 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
}

18.2.
A new tab will be created by the following command:

edm::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 [edm::addTab Service 0]

# Create a button in the center of the bottom of the tab with the # name "OK".
bbutton $name.button -text "OK" -command {w 0x1017 0 u16 500} 
pack $name.button -side bottom -anchor center
18.3.
Tab-set can be deleted with the command:

```
edm::deleteTab (<pos>)
```

**Description**
deletes an additional tab

**Parameters:**
- `pos` position of the tab in the display (starting with 0)

**Return:**
nothing
19.
The Console can be activated via View → Console
In the console Tcl commands can be executed as well as user scripts and procedures.

Figure 38: Console

19.1.
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:

```tcl
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
puts "Bitrate: $myarray(baud)" ;#reference of the array myarray
readWrite
readWrite
exec cdm::pause 100
```

Further information can be found at www.tcl.tk (http://www.tcl.tk).
19.2.

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 EtherCAT® 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:

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

load the script with the source command in the Console:

```con
source example.tcl
```

execute the defined procedure in the Console interactively:

```con
$ showHello Heinz
Hello Heinz
How are you?
```
20.
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 appendant license has been purchased.

20.1.

![DSP 402 State Machine Diagram](image)

**Figure 45:** DSP402 state machine
The state deposited green is the current state of the drive. Pale brown fields indicate pos-
sible next states and gray fields aren’t obtainable directly from the current state. The cur-
rent 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 access mode to the device and the axle of the device. These settings are also valid for the other DSP402 extensions.

20.2.

The profile velocity mode tab simplifies controlling EtherCAT® drives in the profile velocity mode.

![Profile velocity mode tab](image)

**Figure 46**: Profile velocity mode tab

20.3.

The Position Mode Tab simplifies the test and the commissioning 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.
**Figure 47**: DSP402 Position Mode

20.4.

**Figure 48**: Status word bit box
Figure 46: Control word bit box

These extensions are special bit boxes for the status and the control word.
21.

21.1.
The about dialog provides information about:
   • the current release
   • the type of this release
   • the licensee,
   • the license and
   • the available Tcl/Tk packages.

21.2.
When requesting information about the latest release of the EtherCAT® 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.
22.

22.1.

\texttt{r <index> <subindex> <typ>}

\textbf{Description:}
SDO read procedure

\textbf{Parameters:}
- \texttt{index} object index
- \texttt{subindex} object subindex
- \texttt{typ} datatype of object <u8|u16|u32|i8|i16|i32|r32|vs>

\textbf{Results:}
- read value

\texttt{rr <index> <subindex> <typ>}

\textbf{Description:}
SDO read procedure: type out the SDO read command and the answer from the device

\textbf{Parameters:}
- \texttt{index} object index
- \texttt{subindex} object subindex
- \texttt{typ} datatype of object

\textbf{Results:}
- read value

\texttt{rrc <index> <subindex> <typ> <ref>}

\textbf{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 std-out.
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
22.2.

::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 length of l

Parameters:

string unformatted 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 unformatted 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 dialog 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
22.3.

::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|i32|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::getDefaultVal ue  <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
::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_message_log

Description:
  This proc deletes the content of the message log

Parameters:
  -

Results:
  -

save_message_log <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:
   -
22.4.

::402::ret

Description:
List of possible return values

Parameters:
-

Results:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>OK</td>
</tr>
<tr>
<td>1</td>
<td>Drive in wrong state</td>
</tr>
<tr>
<td>2</td>
<td>Transition not possible</td>
</tr>
<tr>
<td>3</td>
<td>SDO abort occurred</td>
</tr>
<tr>
<td>4</td>
<td>No setpoint acknowledge</td>
</tr>
</tbody>
</table>

::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
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.
### ::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:**
- state desired CiA-402 state
- delayTime maximal time for state changing in ms

**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
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
Literature

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

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


The BLT Toolkit.
BLT is an extension to the Tk toolkit, adding new widgets, geometry managers, and miscellaneous commands.
23.

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAN</td>
<td>Controller Area Network</td>
</tr>
<tr>
<td>CAL</td>
<td>CAN Application Layer (CANopen base)</td>
</tr>
<tr>
<td>CDM</td>
<td>CANopen Device Monitor</td>
</tr>
<tr>
<td>CiA</td>
<td>CAN in Automation international users and manufacturers group e.V.</td>
</tr>
<tr>
<td>CN</td>
<td>Controlled Node</td>
</tr>
<tr>
<td>COB</td>
<td>Communication Object (CAN Message)</td>
</tr>
<tr>
<td>COB-ID</td>
<td>Communication Object Identifier</td>
</tr>
<tr>
<td>CSDO</td>
<td>Client SDO</td>
</tr>
<tr>
<td>EDM</td>
<td>EtherCAT Device Monitor</td>
</tr>
<tr>
<td>EDS</td>
<td>Electronic Data Sheet</td>
</tr>
<tr>
<td>ESI</td>
<td>EtherCAT Slave Information file</td>
</tr>
<tr>
<td>EMCY</td>
<td>Emergency Object</td>
</tr>
<tr>
<td>EPSG</td>
<td>Ethernet POWERLINK Standardization Group</td>
</tr>
<tr>
<td>ETG</td>
<td>EtherCAT Technology Group</td>
</tr>
<tr>
<td>MN</td>
<td>Managing Node</td>
</tr>
<tr>
<td>NMT</td>
<td>Network Management</td>
</tr>
<tr>
<td>OD</td>
<td>Object Dictionary</td>
</tr>
<tr>
<td>PDO</td>
<td>Process Data Object, unconfirmed service for real time communication</td>
</tr>
<tr>
<td>RPDO</td>
<td>Receive PDO</td>
</tr>
<tr>
<td>RTR</td>
<td>Remote Transmission Request</td>
</tr>
<tr>
<td>PDM</td>
<td>POWERLINK Device Monitor</td>
</tr>
<tr>
<td>PRMS</td>
<td>Problem Report Management System</td>
</tr>
<tr>
<td>SDO</td>
<td>Service Data Object, confirmed data transfer service for parameter data.</td>
</tr>
<tr>
<td>SSDO</td>
<td>Server SDO</td>
</tr>
<tr>
<td>SYNC</td>
<td>Synchronization Object</td>
</tr>
<tr>
<td>Tcl</td>
<td>Tool Command Language (script language)</td>
</tr>
<tr>
<td>TCP/IP</td>
<td>Transmission Control Protocol/Internet Protocol</td>
</tr>
<tr>
<td>TIME</td>
<td>Time Stamp Object</td>
</tr>
<tr>
<td>TPDO</td>
<td>Transmit PDO</td>
</tr>
<tr>
<td>Widget</td>
<td>element of a graphical user interface (e.g. button, entry filed, menu, ...)</td>
</tr>
</tbody>
</table>
24.

- a -
Action
  tab- 35
  tab-set 10

- b -
bithwise representation 37

- c -
commands, Tcl 44

- e -
edm::addTab 42
edm::addTestTab 39
edm::deleteTab 43

- i -
index 10

- n -
NMT tab 20

- o -
Options 17

- p -
PDO 25
preop 20
Preop All 20
Preop Node 20
procedure 45

- s -
Scan Network 20
script load 45
slider 35–36
start 20
Start All 20
Start Node 20
stop 20
strip-chart 33
subindex 10
system requirements 6

- t -
t_start.tcl 38
tab, Action 35
tab, NMT 20
tab-set, Action 10
tabs, user-specific 38
Tcl commands 44
tree 10

- u -
update 41
user-specific tabs 38

- v -
variable global_stop 41
Table of Contents

Introduction ....................................................... 5
Product Overview .................................................. 5
Conventions .......................................................... 6
Support by port ..................................................... 6
Quick Start ......................................................... 7
First program start ............................................... 7
Communication with a EtherCAT® device ...................... 8
Object Dictionary Accesses .................................... 10
Menu Structure .................................................... 12
File ................................................................. 12
Edit ................................................................. 12
View ................................................................. 13
Connection ........................................................... 13
Extras ............................................................... 13
Windows ............................................................... 15
Help ................................................................. 15
Toolbar ............................................................... 16
Options ............................................................... 17
General Settings ................................................... 17
DCF Settings ........................................................ 18
Network Settings .................................................. 18
Color Settings ...................................................... 19
Font Settings ....................................................... 19
NMT commands .................................................... 20
User defined scripts .............................................. 20
Network overview ................................................ 21
Device Information ................................................ 21
Description Tab .................................................... 22
Object Description ............................................... 22
Object Description File ......................................... 22
Overview Tab ....................................................... 24
Index Overview ................................................... 24
PDO Configuration Tab .......................................... 25
Configuration of PDOs for the chart ......................... 25
Process Tab ......................................................... 28
PDO Process Tab .................................................. 29
File over EtherCAT Tab .......................................... 30
ESC Viewer Tab ................................................... 31
EEPROM Viewer Tab ............................................ 32
Using Stripcharts .................................................. 33
Extended object configuration ................................ 35
Slider ............................................................... 35
Slider in a Top Level Window ................................... 36