zenon

by COPA-DATA

zenon driver manual

MODBUS_ENERGY

v.8.10

Contents

1 Welcome to COPA-DATA help

ZENON VIDEO-TUTORIALS

You can find practical examples for project configuration with zenon in our YouTube channel (https://www.copadata.com/tutorial_menu). The tutorials are grouped according to topics and give an initial insight into working with different zenon modules. All tutorials are available in English.

GENERAL HELP

If you cannot find any information you require in this help chapter or can think of anything that you would like added, please send an email to documentation@copadata.com.

PROJECT SUPPORT

You can receive support for any real project you may have from our Support Team, who you can contact via email at support@copadata.com.

LICENSES AND MODULES

If you find that you need other modules or licenses, our staff will be happy to help you. Email sales@copadata.com.

2 MODBUS_ENERGY

The Open Modbus TCP/IP driver supports Sequence of Events for energy projects and in addition (also for non energy projects) communication via a TCP/IP gateway with several slaves connected to the same serial bus.

When using a Modbus gateway, the driver can be configured so that only one TCP connection to the gateway will be established, even with several devices (slaves) behind the gateway.

The driver supports the reading of the event buffer (events) of:

- AREVA MiCOM P125/P126/P127

- COSTRONIC DFB: supports the reading of events from the Schneider Modicon TSX Premium PLC with the "lynxPileHorodatageV3a" function block (DFB) created by Costronic SA (www.costronic.ch (http://www.costronic.ch)).
- GE Multilin F650 and UR series controls
- IEC NPx800
- Schneider SEPAM

3 MODBUS_ENERGY - data sheet

General:	
Driver file name	MODBUS_ENERGY.exe
Driver name	Modbus Energy Treiber
PLC types	All controllers and gateways with Open Modbus TCP/IP support. Sequence of Events only for AREVA MiCOM P126/P127, GE Multilin F650, GE Multilin UR-series, ICE NPI 800, ICE NPID 800, Schneider MODICON TSX (Premium, M340, Quantum) with function block (DFB) from Costronic and Schneider Sepam protection relay.
PLC manufacturer	ABB; GE Automation&Controls; Modbus RTU; Mondial; Schiele; Telemecanique; Schneider; Wago; SE Elektronic; Areva; GE Multilin; ICE; Costronic

Driver supports:	
Protocol	Modbus RTU over TCP
Addressing: Address-based	Address based
Addressing: Name-based	--
Spontaneous communication	X
Polling communication	X
Online browsing	--
Offline browsing	--
Real-time capable	X

Driver supports:	
Blockwrite	X
Modem capable	--
RDA numerical	X
RDA String	--
Hysteresis	X
extended API	X
Supports status bit **WR-SUC**	X
alternative IP address	X

Requirements:	
Hardware PC	--
Software PC	--
Hardware PLC	--
Software PLC	For Schneider MODICON TSX Premium: Function block (DFB) "cosZenonPileHorodatageV2c" from Costronic
Requires v-dll	--

Platforms:	
Operating systems	Windows 10; Windows 7; Windows 8; Windows 8.1; Windows Server 2008 R2; Windows Server 2012; Windows Server 2012 R2; Windows Server 2016

4 Driver history

Date	Driver version	Change
26.03.09	500	Created driver documentation

Date	Driver version	Change
		The first release test has been conducted (GE Multilin F650 and Areva MiCOM P127)
14.10.09	900	Added COSTRONIC DFB and IEC NPx800.
1/7/2010	1100	GE Multilin UR series added
09.06.10	1699	Schneider SEPAM added
15.01.2016	25506	New driver object type *File record*
12/16/2016	33762	Direct redundancy switching in the event of a failure (FS 37567)

DRIVER VERSIONING

The versioning of the drivers was changed with zenon 7.10. There is a cross-version build number as of this version. This is the number in the 4th position of the file version,
For example: **7.10.0.4228** means: The driver is for version **7.10** service pack **0**, and has the build number **4228**.

Expansions or error rectifications will be incorporated into a build in the future and are then available from the next consecutive build number.

Example

A driver extension was implemented in build **4228**. The driver that you are using is build number **8322**. Because the build number of your driver is higher than the build number of the extension, the extension is included. The version number of the driver (the first three digits of the file version) do not have any significance in relation to this. The drivers are version-agnostic

5 Requirements

This chapter contains information on the requirements that are necessary for use of this driver.

5.1 Control

For the use of the Modbus Energy Driver, the following conditions apply:

- The PLC must support the Open Modbus TCP/IP protocol in the conformity class 0
- The reading of the event buffer is only possible with the PLCs mentioned in chapter MODBUS_ENERGY (on page 5).
- The PLCs should be set to UTC time. The driver assumes that all incoming time stamps are in UTC time.

AREVA MICOM P125/P126/P127

The time format in Areva MiCOM P12/P125/P127 Register 012Fh should be set to "Internal format" (not to "IEC format").

COSTRONIC DFB

The block should be carried out by task "FAST" and should be configured as follows.

- **ControleVie**, **IndiceEchange** and **balEvenements** must be behind one another, for example %MW900 and %MW901 and %MW902:56
- **balEvenements** must be 56 registers large
- **NombreMaxEven...** must lie between *1* and *7* (number of events in the mailbox)
- The stack for buffering of events is connected to **pile**
- **tableEvenments** contains the status of all events (1 bit per event)
- **numeroPremierEvenement** must be *0*.

6 Configuration

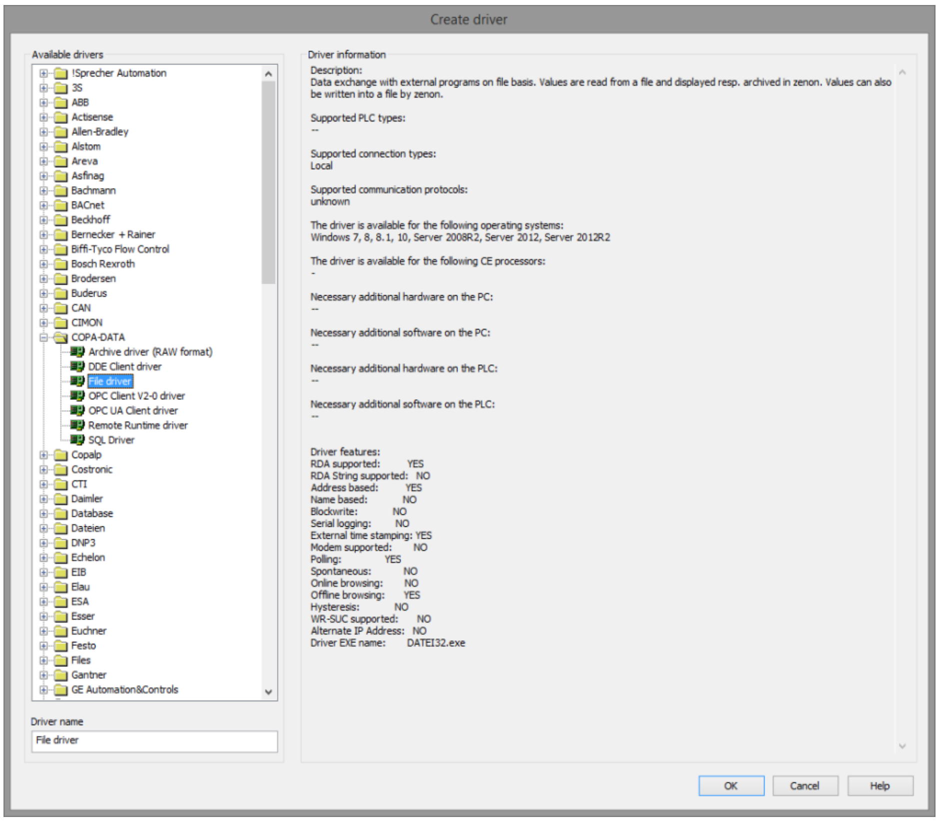
In this chapter you will learn how to use the driver in a project and which settings you can change.

Information

Find out more about further settings for zenon variables in the chapter Variables (main.chm::/15247.htm) of the online manual.

6.1 Creating a driver

In the **Create driver** dialog, you create a list of the new drivers that you want to create.



!Sprecher Automation
3S
ABB
Actisense
Allen-Bradley
Alstom
Areva
Asfinag
Bachmann
BACnet
Beckhoff
Bernecker + Rainer
Biffi-Tyco Flow Control
Bosch Rexroth
Brodersen
Buderus
CAN
CIMON
COPA-DATA
- Archive driver (RAW format)
- DDE Client driver
- File driver
- OPC Client V2-0 driver
- OPC UA Client driver
- Remote Runtime driver
- SQL Driver

Copalp
Costronic
CTI
Daimler
Database
Dateien
DNP3
Echelon
EIB
Elau
ESA
Esser
Euchner
Festo
Files
Gantner
GE Automation&Controls

Driver name: File driver

```
Description:
Data exchange with external programs on file basis. Values are read from a file and displayed resp. archived in zenon. Values can also
be written into a file by zenon.

Supported PLC types:
--

Supported connection types:
Local

Supported communication protocols:
unknown

The driver is available for the following operating systems:
Windows 7, 8, 8.1, 10, Server 2008R2, Server 2012, Server 2012R2

The driver is available for the following CE processors:
-

Necessary additional hardware on the PC:
--

Necessary additional software on the PC:
--

Necessary additional hardware on the PLC:
--

Necessary additional software on the PLC:
--


Driver features:
RDA supported:         YES
RDA String supported:  NO
Address based:         YES
Name based:            NO
Blockwrite:            NO
Serial logging:        NO
External time stamping: YES
Modem supported:       NO
Polling:               YES
Spontaneous:           NO
Online browsing:       NO
Offline browsing:      YES
Hysteresis:            NO
WR-SUC supported:      NO
Alternate IP Address:  NO
Driver EXE name:       DATEI32.exe
```

Parameter	Description
Available drivers	List of all available drivers. The display is in a tree structure: *[+]* expands the folder structure and shows the drivers contained therein. [-] reduces the folder structure Default: *no selection*
Driver name	Unique **Identification** of the driver. Default: *Empty* The input field is pre-filled with the pre-defined **Identification** after selecting a driver from the list of available drivers.
Driver information	Further information on the selected driver. Default: *Empty* The information on the selected driver is shown in this area after selecting a driver.

CLOSE DIALOG

Option	Description
OK	Accepts all settings and opens the driver configuration dialog of the selected driver.
Cancel	Discards all changes and closes the dialog.
Help	Opens online help.

Information

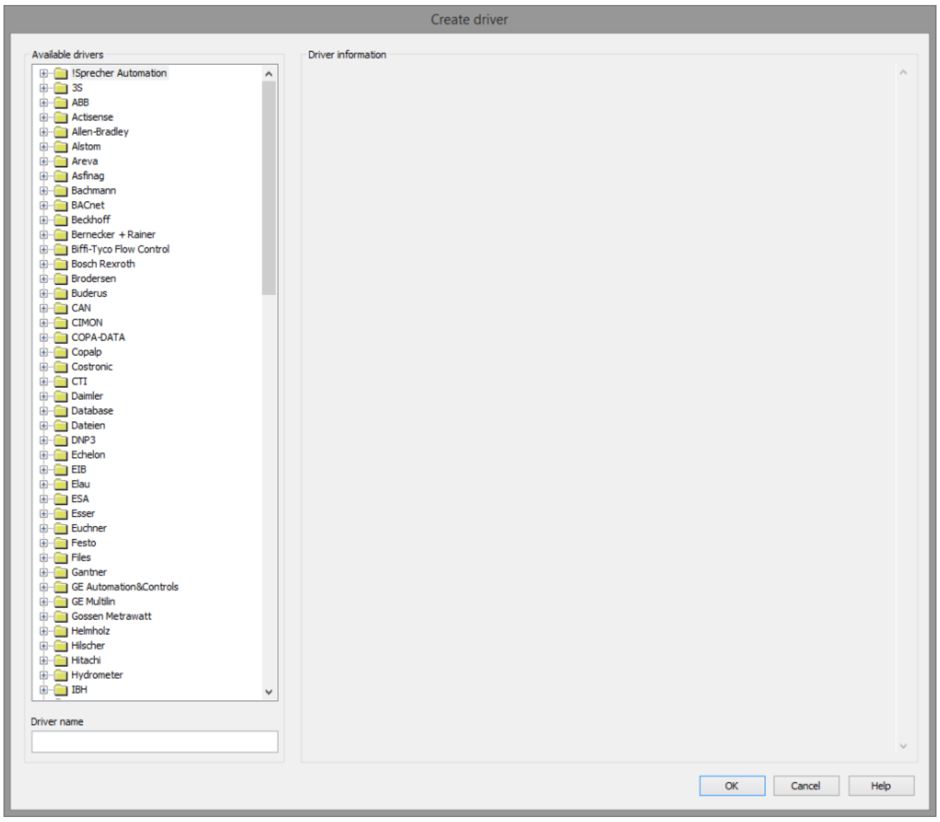
The content of this dialog is saved in the file called Treiber_[Language].xml. You can find this file in the following folder: *C:\ProgramData\COPA-DATA\zenon[version number]*.

CREATE NEW DRIVER

In order to create a new driver:

1. Right-click on **Driver** in the Project Manager and select **New driver** in the context menu. Optional: Select the **New driver** button from the toolbar of the detail view of the **Variables**. The **Create driver** dialog is opened.

2. The dialog offers a list of all available drivers.



3. Select the desired driver and name it in the **Driver name** input field.
   This input field corresponds to the **Identification** property. The name of the selected driver is automatically inserted into this input field by default.
   The following is applicable for the **Driver name**:

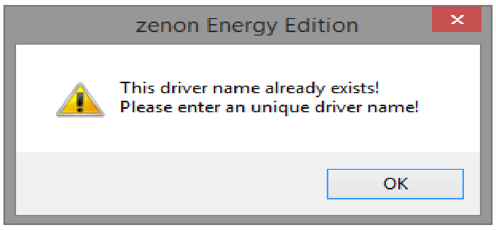
   - The **Driver name** must be unique.
     If a driver is used more than once in a project, a new name has to be given each time.
     This is evaluated by clicking on the **OK** button. If the driver is already present in the project, this is shown with a warning dialog.
   - The **Driver name** is part of the file name.
     Therefore it may only contain characters which are supported by the operating system.
     Invalid characters are replaced by an underscore (_).
   - **Attention:** This name cannot be changed later on.

4. Confirm the dialog by clicking on the **OK** button.
   The configuration dialog for the selected driver is opened.

Note: The language of driver names cannot be switched. They are always shown in the language in which they have been created, regardless of the language of the Editor. This also applies to driver object types.

DRIVER NAME DIALOG ALREADY EXISTS

If there is already a driver in the project, this is shown in a dialog. The warning dialog is closed by clicking on the **OK** button. The driver can be named correctly.


zenon Energy Edition

This driver name already exists!
Please enter an unique driver name!

ZENON PROJECT

The following drivers are created automatically for newly-created projects:

- **Intern**
- **MathDr32**
- **SysDrv**

Information

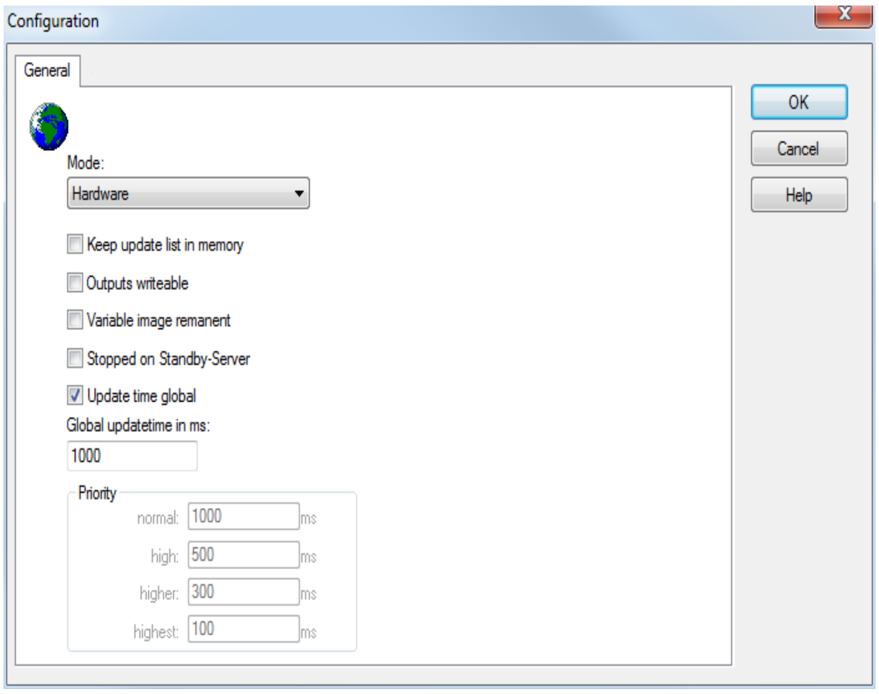
Only the required drivers need to be present in a zenon project. Drivers can be added at a later time if required.

6.2 Settings in the driver dialog

You can change the following settings of the driver:

6.2.1 General

The configuration dialog is opened when a driver is created. In order to be able to open the dialog later for editing, double click on the driver in the list or click on the **Configuration** property.


Configuration

General

Mode: Hardware

- [ ] Keep update list in memory
- [ ] Outputs writeable
- [ ] Variable image remanent
- [ ] Stopped on Standby-Server
- [x] Update time global

Global updatetime in ms: 1000

Priority

normal: 1000 ms

high: 500 ms

higher: 300 ms

highest: 100 ms

Option	Description
Mode	Allows to switch between hardware mode and simulation mode ▶ *Hardware*: A connection to the control is established. ▶ Simulation - static: No communication between to the control is established, the values are simulated by the driver. In this modus the values remain constant or the variables keep the values which were set by zenon Logic. Each variable has its own memory area. E.g. two variables of the type marker with offset 79 can have different values in the Runtime and do not influence each other. Exception: The simulator driver. ▶ *Simulation - counting*: No communication between to the control is established, the values are simulated by the driver. In this modus the driver increments the values

Option	Description
	within a value range automatically. - *Simulation - programmed*: No communication is established to the PLC. The values are calculated by a freely programmable simulation project. The simulation project is created with the help of the zenon Logic Workbench and runs in a zenon Logic Runtime which is integrated in the driver. For details see chapter Driver simulation (main.chm::/25206.htm).
Keep update list in the memory	Variables which were requested once are still requested from the control even if they are currently not needed. This has the advantage that e.g. multiple screen switches after the screen was opened for the first time are executed faster because the variables need not be requested again. The disadvantage is a higher load for the communication to the control.
Output can be written	- *Active*: Outputs can be written. - *Inactive*: Writing of outputs is prevented. **Note**: Not available for every driver.
Variable image remanent	This option saves and restores the current value, time stamp and the states of a data point. Fundamental requirement: The variable must have a valid value and time stamp. The variable image is saved in hardware mode if one of these statuses is active: - User status *M1* (*0*) to *M8* (*7*) - *REVISION(9)* - *AUS(20)* - *ERSATZWERT(27)* The variable image is always saved if: - the variable is of the object type **Driver variable** - the driver runs in simulation mode. (not

Option	Description
	programmed simulation) The following states are not restored at the start of the Runtime: - *SELECT(8)* - *WR-ACK(40)* - *WR-SUC(41)* The mode **Simulation - programmed** at the driver start is not a criterion in order to restore the remanent variable image.
Stop on Standby Server	Setting for redundancy at drivers which allow only one communication connection. For this the driver is stopped at the Standby Server and only started at the upgrade. **Attention:** If this option is active, the gapless archiving is no longer guaranteed. - *Active*: Sets the driver at the not-process-leading Server automatically in a stop-like state. In contrast to stopping via driver command, the variable does not receive status **switched off** (**statusverarbeitung.chm::/24150.htm**) but an empty value. This prevents that at the upgrade to the Server irrelevant values are created in the AML, CEL and Historian. Default: *inactive* **Note:** Not available if the CE terminal serves as a data server. You can find further information in the zenon Operator manual in the CE terminal as a data server chapter.
Global Update time	Setting for the global update times in milliseconds: - *Active*: The set **Global update time** is used for all variables in the project. The priority set at the variables is not used. - *Inactive*: The set priorities are used for the individual variables.

Option	Description
	Exceptions: Spontaneous drivers ignore this option. They generally use the shortest possible update time. For details, see the **Spontaneous driver update time** section.
Priority	The polling times for the individual priority classes are set here. All variables with the according priority are polled in the set time. The variables are allocated separately in the settings of the variable properties. The communication of the individual variables can be graded according to importance or required topicality using the priority classes. Thus the communication load is distributed better. **Attention:** Priority classes are not supported by each driver, e.g. spontaneously communicating zenon drivers.

CLOSE DIALOG

Option	Description
OK	**Applies all changes in all tabs and closes the dialog.**
Cancel	Discards all changes in all tabs and closes the dialog.
Help	Opens online help.

UPDATE TIME FOR SPONTANEOUS DRIVERS

With spontaneous drivers, for **Set value**, **advising** of variables and **Requests**, a read cycle is triggered immediately - regardless of the set update time. This ensures that the value is immediately available for visualization after writing. The update time is generally 100 ms.

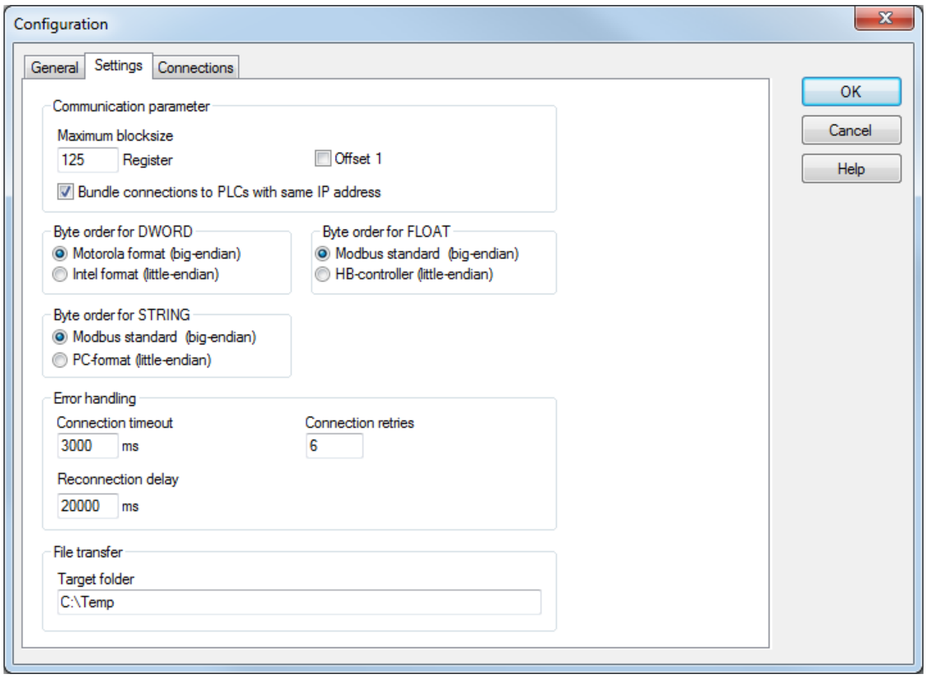
Spontaneous drivers are **ArchDrv**, **BiffiDCM**, **BrTcp32**, **DNP3**, **Esser32**, **FipDrv32**, **FpcDrv32**, **IEC850**, **IEC870**, **IEC870_103**, **Otis**, **RTK9000**, **S7DCOS**, **SAIA_Slave**, **STRATON32** and **Trend32**.

6.2.2 Settings

You can configure the general settings for all Open Modbus TCP/IP connections in the tab **Settings**. To do this:

- click on the property **Configuration** in the group **General**

- now the dialog for the configuration of the driver opens.
- select the Settings tab



Communication parameter
Maximum blocksize
125 Register
☐ Offset 1
☑ Bundle connections to PLCs with same IP address

Byte order for DWORD
● Motorola format (big-endian)
○ Intel format (little-endian)

Byte order for FLOAT
● Modbus standard (big-endian)
○ HB-controller (little-endian)

Byte order for STRING
● Modbus standard (big-endian)
○ PC-format (little-endian)

Error handling
Connection timeout
3000 ms
Connection retries
6
Reconnection delay
20000 ms

File transfer
Target folder
C:\Temp

COMMUNICATION PARAMETER

Parameter	Description
Maximum block size	Defines the maximum number of registers that can be queried or written using a data telegram. The configured value applies for both read and write requests. If you configure a value bigger than *100*, a maximum number of 100 registers will be used for write requests. Allowed value range: *1* - *125*.
Offset 1	*Active*: The driver subtracts *1* when the variable addresses (coils, register) are sent and adds *1* when they are received. Affects the addressing of all variables.
Grouping connections to PLCs with the same IP address	*Active*: Connections with the same IP address and port number will be grouped. If you want to establish only one TCP connection to a gateway, activate this option.

BYTE SEQUENCE FOR DWORD

Parameter	Description
Byte sequence for DWORD	Defines the sequence of lower-value and higher-value words for double word objects (DINT/UDINT). You can choose between Motorola (Big-Endian) and Intel (Little-Endian).
Motorola Format (Big-Endian)	*Active*: DWORD ordering in accordance with Motorola format.
Intel Format (Little-Endian)	*Active*: DWORD ordering in accordance with Intel format.

BYTE SEQUENCE FLOAT

Parameter	Description
Byte sequence FLOAT	Defines the sequence of lower-value and higher-value words for FLOAT objects (REAL). You can choose between Modbus Standard (Big-Endian) and HB-Controller (Little-Endian).
Modbus Standard (Big-Endian)	*Active*: Float ordering in accordance with Modbus standard.
HB Controller Float (Little-Endian)	*Active*: Float ordering in accordance with HB controller.

BYTE SEQUENCE FOR STRING

Parameter	Description
Byte sequence for STRING	Defines display of the byte sequence. **Note:** If there is relevant information in the documentation of a PLC the following is usually applicable: MSB-first = ▶ Motorola = Big-Endian = Modbus. ▶ Intel = Little-Endian = PC. This can be handled differently in some documentation however.
Modbus Standard (Big-Endian)	Display in accordance with Modbus standard with switched characters.
PC-Format (Little-Endian)	Display in accordance with Modbus standard with switched

Parameter	Description
	characters.

ERROR HANDLING

These settings are relevant for redundancy switching in the event of a failure of a connection.

Parameter	Description
Connection timeout	Time in milliseconds to wait for a response from the Slave. A communication error will be displayed if there is no response within this time. Default: *3000 ms*
Retries on error	Number of send repetitions when there is no answer from the slave after the set communication timeout. ▸ *1*: a connection attempt, no repetitions ▸ *0*: constant repetition Default: *6*
Delay after connection termination	Time in milliseconds to wait after a communication error before trying to reestablish the connection. Default: *20000 ms*
Target folder	Folder for file transfer.

CLOSE DIALOG

Option	Description
OK	**Applies all changes in all tabs and closes the dialog.**
Cancel	Discards all changes in all tabs and closes the dialog.
Help	Opens online help.

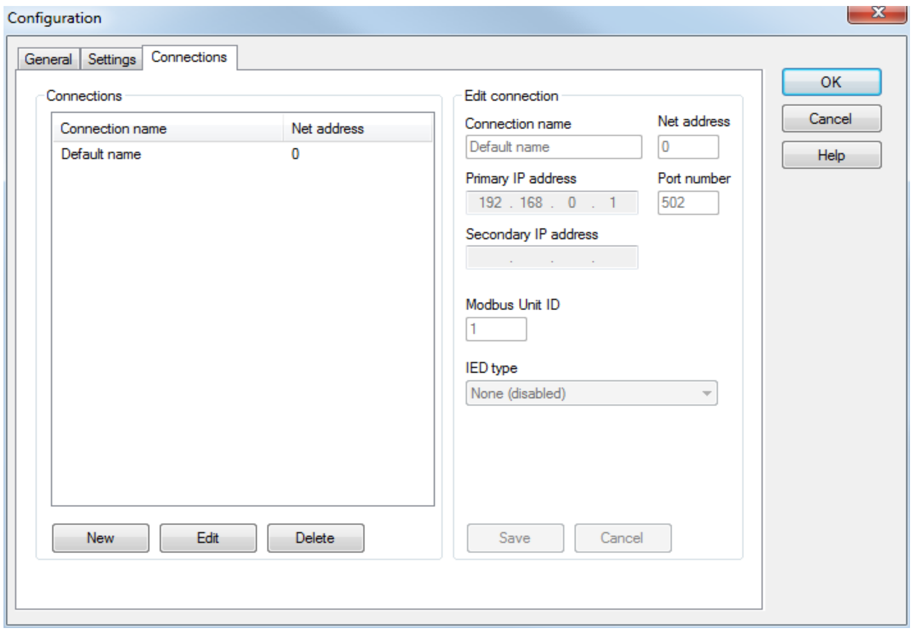
Information

Sequents of Events: If a variable is created for a connection in the event area, SOE is used. Otherwise SOE is not used.

6.2.3 Connections

You can configure the connections to the PLCs via Open Modbus TCP/IP in the tab **Connections**: To do this:

- click on the property **Configuration** in the group **General**
- now the dialog for the configuration of the driver opens.
- select the tab **Connections**



Parameter	Description
Connection list	Displays the connection names with the corresponding hardware addresses. To display the connection parameters of a name, click on the according connection name.
Net address	The net address identifies the connection. Therefore, every connection must have a unique net address. Variables are assigned to a connection via the net address. At the creation of a new variable the net address with the highest existing address + 1 is initialized.
Connection name	Freely definable name for the easier distinction of connections. **Attention:** The connection name must not contain any of the following characters: *{}|&~![]()"';=*
Primary IP address	Primary IP address of the PLC with which you are communicating.
Port number	The port number of the PLC.

Parameter	Description
	Default for Open Modbus TCP/IP: 502
Secondary IP address	Secondary IP address of the PLC with which you are communicating. If the connection to the **primary address** fails, an attempt to connect is made using this. The secondary connection is used until a failure or restart; or switching with the *SwitchConnection* command. **Note:** in the event of a connection failure, switching is carried out after expiry of the configured communication timeout times.
Modbus Unit ID	*Modbus Unit ID* (slave address) - the address of the recipient.
IED Typ	Selection of the IED type (IED = Intelligent Electronic Device). Select from drop-down list: - *AREVA MiCOM P126/127* - *COSTRONIC DFB* - *GE Multilin F650* - *GE Multilin UR-series* - *ICE NPx800* - *Schneider SEPAM* - *None (disabled) - Standard Modbus TCP/IP* Default: *None (disabled)*
New	Creates a new connection with default settings.
Delete	Deletes the connection selected in the connection list.
Edit	Opens the selected connection for editing.
Save	Saves a new or edited connection.
Cancel	Cancels the editing of connection settings without saving changes.

ENHANCEMENT WHEN SELECTING COSTRONIC DFB

Parameter	Description
Base register Mailbox	The address of *ControleVie*.
Base register Events	Base address of *tableEvenments*.
Number of registers	Length of the array which is connected to *tableEvenments* (number

Parameter	Description
	of the Event bit/16).

CREATE NEW CONNECTION

1. click on the button **New**
2. enter the connection parameters **Net address**, **Connection name** and **Modbus Unit ID**
3. Choose the **IED type** if necessary
4. Click on **Save**

Note: If a established connection is selected at the creation, its configuration is copies.

DISPLAY CONNECTION PARAMETERS OF A CONNECTION

1. select the desired connection in the connection list with the mouse pointer.
2. the parameters will be displayed

EDIT CONNECTION

1. select the connection in the connection list
2. Click on the **Edit** button
3. change the connection parameters
4. finish with **Save**

DELETE CONNECTION

1. select the connection in the connection list
2. click on the button **Delete**
3. the connection will be removed from the list

7 Creating variables

This is how you can create variables in the zenon Editor:

7.1 Creating variables in the Editor

Variables can be created:

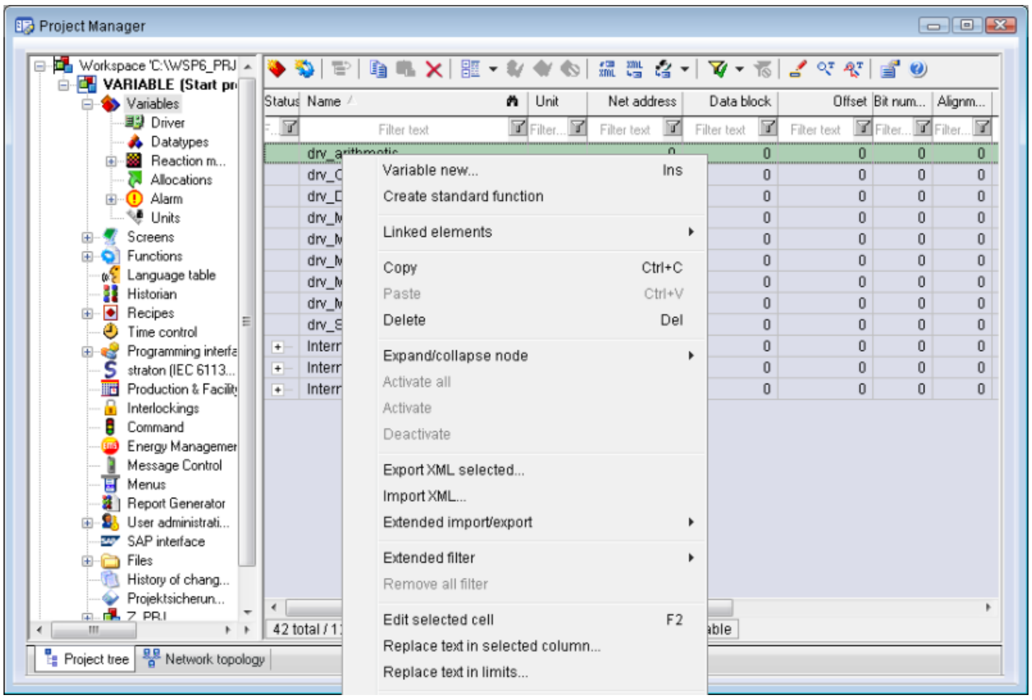
- as simple variables

- in arrays (main.chm::/15262.htm)
- as structure variables (main.chm::/15278.htm)

VARIABLE DIALOG

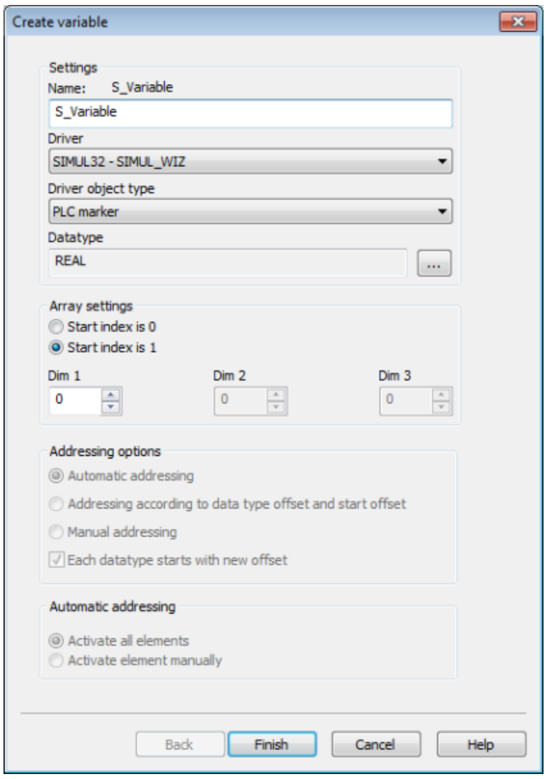
To create a new variable, regardless of which type:

1. Select the **New variable** command in the **Variables** node in the context menu



   The dialog for configuring variables is opened
2. Configure the variable
3. The settings that are possible depends on the type of variables

CREATE VARIABLE DIALOG


Create variable

Settings

Name: S_Variable

S_Variable

Driver: SIMUL32 - SIMUL_WIZ

Driver object type: PLC marker

Datatype: REAL

Array settings

Start index is 0 [✓ Start index is 1]

Dim 1: 0 | Dim 2: 0 | Dim 3: 0

Addressing options

[✓ Automatic addressing]

Addressing according to data type offset and start offset

Manual addressing

[✓ Each datatype starts with new offset]

Automatic addressing

[✓ Activate all elements]

Activate element manually

Property	Description
Name	Distinct name of the variable. If a variable with the same name already exists in the project, no additional variable can be created with this name. Maximum length: 128 characters **Attention:** the characters **#** and **@** are not permitted in variable names. If non-permitted characters are used, creation of variables cannot be completed and the **Finish** button remains inactive. **Note:** For some drivers, the addressing is possible over the property **Symbolic address**, as well.
Drivers	Select the desired driver from the drop-down list. **Note:** If no driver has been opened in the project, the driver for internal variables (**Intern.exe** (**Main.chm::/Intern.chm::/Intern.htm**)) is automatically loaded.
Driver Object Type (**cti.chm::/28685.htm**)	Select the appropriate driver object type from the drop-down list.
Data Type	Select the desired data type. Click on the ... button to open the

Property	Description
	selection dialog.
Array settings	Expanded settings for array variables. You can find details in the Arrays chapter.
Addressing options	Expanded settings for arrays and structure variables. You can find details in the respective section.
Automatic addressing	Expanded settings for arrays and structure variables. You can find details in the respective section.

SYMBOLIC ADDRESS

The **Symbolic address** property can be used for addressing as an alternative to the **Name** or **Identification** of the variables. Selection is made in the driver dialog; configuration is carried out in the variable property. When importing variables of supported drivers, the property is entered automatically.

Maximum length: *1024* characters.

The following drivers support the **Symbolic address**:

- 3S_V3
- AzureDrv
- BACnetNG
- IEC850
- KabaDPServer
- OPCUA32
- Phoenix32
- POZYTON
- RemoteRT
- S7TIA
- SEL
- SnmpNg32
- PA_Drv

INHERITANCE FROM DATA TYPE

Measuring range, **Signal range** and **Set value** are always:

- derived from the datatype
- Automatically adapted if the data type is changed

Note for signal range: If a change is made to a data type that does not support the set **signal range**, the **signal range** is amended automatically. For example, for a change from **INT** to **SINT**, the **signal range** is changed to *127*. The amendment is also carried out if the **signal range** was not inherited from the data type. In this case, the **measuring range** must be adapted manually.

7.2 Addressing

You define the addressing of the variables in the property window:

Group/Property	Description
General	Property group for general settings.
Name	Freely definable name. **Attention:** For every zenon project the name must be unambiguous.
Identification	Freely definable identification. E.g. for Resources label, comments, ...
Addressing	
Net address	This address refers to the bus address in the connection configuration of the driver. Specifies, on which PLC the variable resides.
Data block	not used for this driver
Offset	The meaning is dependent on the setting of the driver object types. - For variables of the driver object type **Register** or **SOE - Register event**: Reference number of the register [*0* to *65535*] - For variables of the driver object type **SOE - Numbered event**: Identifies the event. Value range depends on control unit, for example: - AREVA MiCOM P125/P126/P127: *1* to *127 o.162* - Costronic DFB*: 0* to *(number of registers)*bit-1* Example with 63 registers: **63*16-1**=**1007**; range **0** to **1007** - GE Multilin F650: *0* to *191* - GE Multilin UR: *0* to number of FlexOperandStates - ICE NPx800: to *8000h* - Schneider SEPAM: *1000h* to *105Fh*
Alignment	Alignment for variables with byte length *1*. You can choose between low byte and high byte.
Bit number	Number of the bit within the configured offset. Possible entries: *0* ... *15*

Group/Property	Description
String length	Only available for String variables. Maximum number of characters that the variable can take.
Driver connection/Driver Object Type	Allows you to change the driver object type that was selected when the variable was created.
Driver connection/Data Type	Data type of the variable. Is selected during the creation of the variable; the type can be changed here. **Attention:** If you change the data type later, all other properties of the variable must be checked and adjusted, if necessary.
Driver connection/Priority	Setting the priority class. The variable of the priority class is thus assigned as it was configured in the driver dialog in the **General** tab. The priority classes are only used if the **global update time** is deactivated. If the **global update time** option is activated and the priority classes are used, there is an error entry in the log file of the system. The driver uses the highest possible priority.

7.3 Driver objects and datatypes

Driver objects are areas available in the PLC, such as markers, data blocks etc. Here you can find out which driver objects are provided by the driver and which IEC data types can be assigned to the respective driver objects.

7.3.1 Driver objects

The following object types are available in this driver:

Driver-object type	Channel type	Read: Modbus function (Code hex/dec)	Write: Modbus function (Code hex/dec)	Supported data types	Comment
Register	8	0x03/3	0x10/16	*BOOL, DINT, UDINT, INT, UINT, LINT, ULINT, REAL, LREAL, SINT,*	Class 0 - multiple registers. Linear addressing: ‣ Bit: one-step via

Driver-object type	Channel type	Read: Modbus function (Code hex/dec)	Write: Modbus function (Code hex/dec)	Supported data types	Comment
				USINT, STRING	**offset** and **bit number** - Byte (8 bits): one-step via **Offset** and **Orientation** - Word (16 bits) Double word (32 bits) Float (32 bits) String(n*Byte): one-step via **Offset**
File record	68	0x14	0x15	*String*	Addressing: The string is addressed using the net address (connection number), the data block (file record number) and the offset (number of the record in the file record). **Note:** Note the information in the file record (on page 33) chapter.
Coil	65	0x01/1	0x05/5	*BOOL*	Class 1 - coils. Linear addressing one-step via **offset**.
File transfer	12	X	X	*STRING*	PLC specific.
Input discrete	66	0x02/2	N/A	*BOOL*	Class 1 - input discretes. Linear addressing one-step via **offset**.

Driver-object type	Channel type	Read: Modbus function (Code hex/dec)	Write: Modbus function (Code hex/dec)	Supported data types	Comment
Input register	64	0x04/4	N/A	*BOOL, SINT, USINT, INT, UINT, DINT, UDINT, REAL, LREAL, STRING*	Class 1 - input registers. Linear addressing as with *registers*.
SOE - Numbered vent	10	X	N/A	*BOOL, STRING*	PLC specific. Sequence of Events (read event buffer): Addressing of events takes place according to event number.
SOE - Register event	11	X	N/A	*BOOL, USINT, UINT*	PLC specific. Sequence of Events (read event buffer): Addressing according to the address of the corresponding Modbus register allocated to the event. Not suitable for COSTRONIC.
Status	67	X	N/A	*UINT, INT*	Status variables. For more details see chapter Driver-specific functions (on page 48).
Communication details	35	X	X	*BOOL, SINT, USINT, INT, UINT, DINT, UDINT, REAL, STRING*	Variables for the static analysis of the communication; is transferred between driver and Runtime (not to the PLC).

Driver-object type	Channel type	Read: Modbus function (Code hex/dec)	Write: Modbus function (Code hex/dec)	Supported data types	Comment
					Note: The addressing and the behavior is the same for most zenon drivers. You can find detailed information on this in the Communication details (Driver variables) (on page 42) chapter.

Key:

X: supported

--: not supported

Attention

Note:
If values are written to a string variable in zenon, with a configured **String length** of approximately 130 characters, only one character length of 129 characters is transferred to the PLC. The last character is replaced with a zero terminator.

7.3.1.1 Function codes

MODBUS FUNCTION CODES

Function code hex/dec	Modbus identifier	Comment
0x01 / 1	Read coils	This function code is used to read from 1 to 2000 contiguous status of coils (bits) in a remote device

Function code hex/dec	Modbus identifier	Comment
0x02/2	Read discrete inputs	This function code is used to read from 1 to 2000 contiguous status of discrete inputs (bits) in a remote device
0x03 / 3	Read multiple registers	This function code is used to read a block of contiguous holding registers (1 to 125 words) in a remote device.
0x04 / 4	Read input registers	This function code is used to read a block of contiguous input registers (1 to 125 words) in a remote device.
0x05 / 5	Write coil	This function code is used to write a single output (one bit) to either ON or OFF in a remote device.
0x06 / 6	Write single register	This function code is used to write a single (one word) holding register in a remote device.
0x10 / 16	Write multiple registers	This function code is used to write a block of contiguous holding registers (1 to approx. 120 words) in a remote device.
0x11 / 17	Report Slave ID	This function code is used to read the description of the type, the current status, and other information specific to a remote device.
0x14/20	Read File Record	This function code is used to perform a file record read. All Request Data Lengths are provided in terms of number of bytes and all Record Lengths are provided in terms of registers. **Note:** This Functioncode is supported by the MODBUS_ENERGY driver only.
0x15/21	Write File Record	This function code is used to perform a file record write. All Request Data Lengths are provided in terms of number of bytes and all Record Lengths are provided in terms of the number of 16-bit words. **Note:** This Functioncode is supported by the MODBUS_ENERGY driver only.

Function code hex/dec	Modbus identifier	Comment
0x2B/43	Read Device Identification	This function code allows reading the identification and additional information relative to the physical and functional description of a remote device.

7.3.1.2 File Record

READ AND WRITE ACCESS

The content of a „*File Record*" is provided by the driver as a hexadecimal string when writing or received as such for writing:

- If the „*File Record*" contains the string "12345", the driver will forward this to the application as a hexadecimal string "3132333435".
- If the „*File Record*" contains the string "abcde", the application must pass the string "6162636465" on to the driver.

DRIVER OBJECT TYPE

The string is addressed using the net address (connection number), the data block (file record number) and the offset (number of the record in the file record).

READ VARIABLE

File record variables are read as polling.

It is always the defined length of the string that is read by the PLC:

- With a given string length of 10, 5 bytes are read from the PLC, coded as a hexadecimal string (thus resulting in the length of 10) and forwarded to the application.
- If the string length is an uneven number, it is rounded up for the reading of data by the PLC, but the lower-value nibble of the last byte is discarded.

WRITING VARIABLES

When writing, the length of the string passed on to the driver determines the amount of data that is actually written.

It must be noted that the smallest data unit that can be written is one word – 2 bytes or 16 bits. The string length should therefore always be a multiple of 4. If this is not the case, excess data bytes in the string are ignored!

It must also be noted that the string must be coded in hexadecimal. Only the figures 0 to 9 (values 0 to 9) and the letters A to F (values 10 to 15) are permitted. The characters can be written with small letters or capital letters. If a character that cannot be used is detected during conversion, there is no write access to the PLC.

CONSTANT

Constant	Value	Description
MB_MIN_FREC_FILENO	*1*	Lowest valid file number
MB_MAX_FREC_FILENO	*65535*	Highest valid file number
MB_MIN_FREC_RECNO	*0*	Lowest valid record number
MB_MAX_FREC_RECNO	*65535*	Highest valid record number
MB_FREC_MINSTRLEN	*4*	Minimum string length – this value must not be gone below if data is to be written.
MB_FREC_MAXSTRLEN	*476*	Maximum string length

7.3.2 Mapping of the data types

All variables in zenon are derived from IEC data types. The following table compares the IEC datatypes with the datatypes of the PLC.

PLC	zenon	Data type
Register / Event	BOOL	8
Register	USINT	9
Register	SINT	10
Register	UINT	2
Register	INT	1
Register	UDINT	4
Register	DINT	3
Register	ULINT	27

PLC	zenon	Data type
Register	LINT	26
Register	REAL	5
Register	LREAL	6
Register	STRING	12
-	WSTRING	21
-	DATE	18
-	TIME	17
-	DATE_AND_TIME	20
-	TOD (Time of Day)	19

DATA TYPE

The term **data type** is the internal numerical identification of the data type. It is also used for the extended DBF import/export of the variables.

7.4 Creating variables by importing

Variables can also be imported by importing them. The XML and DBF import is available for every driver.

Information

You can find details on the import and export of variables in the Import-Export (main.chm::/13028.htm) manual in the Variables (main.chm::/13045.htm) section.

7.4.1 XML import

During XML import of variables or data types, these are first assigned to a driver and then analyzed. Before import, the user decides whether and how the respective element (variable or data type) is to be imported:

- *Import*:
  The element is imported as a new element.

- *Overwrite*:
  The element is imported and overwrites a pre-existing element.
- *Do not import*:
  The element is not imported.

Note: The actions and their durations are shown in a progress bar during import. The import of variables is described in the following documentation. Data types are imported along the same lines.

REQUIREMENTS

The following conditions are applicable during import:

- **Backward compatibility**

  At the XML import/export there is no backward compatibility. Data from older zenon versions can be taken over. The handover of data from newer to older versions is not supported.

- **Consistency**

  The XML file to be imported has to be consistent. There is no plausibility check on importing the file. If there are errors in the import file, this can lead to undesirable effects in the project.

  Particular attention must be paid to this, primarily if not all properties exist in the XML file and these are then filled with default values. E.g.: A binary variable has a limit value of *300*.

- **Structure data types**

  Structure data types must have the same number of structure elements.
  Example: A structure data type in the project has 3 structure elements. A data type with the same name in the XML file has 4 structure elements. Then none of the variables based on this data type in the file are imported into the project.

Hint

You can find further information on XML import in the **Import - Export** manual, in the **XML import** (**main.chm::/13046.htm**) chapter.

7.4.2 DBF Import/Export

Data can be exported to and imported from dBase.

Information

Import and Export via CSV or dBase supported; no driver specific variable settings, such as formulas. Use export/import via XML for this.

IMPORT DBF FILE

To start the import:

1. right-click on the variable list
2. in the drop-down list of **Extended export/import...** select the **Import dBase** command
3. follow the import assistant

The format of the file is described in the chapter File structure.

Information

Note:

- Driver object type and data type must be amended to the target driver in the DBF file in order for variables to be imported.
- dBase does not support structures or arrays (complex variables) at import.

EXPORT DBF FILE

To start the export:

1. right-click on the variable list
2. in the drop-down list of **Extended export/import...** select the **Export dBase...** command
3. follow the export assistant

Attention

DBF files:

- must correspond to the 8.3 DOS format for filenames (8 alphanumeric characters for name, 3 character suffix, no spaces)
- must not have dots (.) in the path name.
  e.g. the path *C:\users\John.Smith\test.dbf* is invalid.
  Valid: *C:\users\JohnSmith\test.dbf*
- must be stored close to the root directory in order to fulfill the limit for file name length including path: maximum 255 characters

The format of the file is described in the chapter File structure.

Information

dBase does not support structures or arrays (complex variables) at export.

FILE STRUCTURE OF THE DBASE EXPORT FILE

The dBaseIV file must have the following structure and contents for variable import and export:

Attention

dBase does not support structures or arrays (complex variables) at export.

DBF files must:

- conform with their name to the 8.3 DOS format (8 alphanumeric characters for name, 3 characters for extension, no space)
- Be stored close to the root directory (Root)

STRUCTURE

Identification	Type	Field size	Comment
KANALNAME	Char	128	Variable name. The length can be limited using the **MAX_LAENGE** entry in the **project.ini** file.
KANAL_R	C	128	The original name of a variable that is to be replaced by the new name entered under "VARIABLENNAME" (variable name) (field/column must be entered manually). The length can be limited using the **MAX_LAENGE** entry in the **project.ini** file.
KANAL_D	Log	1	The variable is deleted with the *1* entry (field/column has to be created by hand).
TAGNR	C	128	Identification. The length can be limited using the **MAX_LAENGE** entry in the **project.ini** file.
EINHEIT	C	11	Technical unit
DATENART	C	3	Data type (e.g. bit, byte, word, ...) corresponds to the

Identification	Typ e	Field size	Comment
			data type.
KANALTYP	C	3	Memory area in the PLC (e.g. marker area, data area, ...) corresponds to the driver object type.
HWKANAL	Nu m	3	Net address
BAUSTEIN	N	3	Datablock address (only for variables from the data area of the PLC)
ADRESSE	N	5	Offset
BITADR	N	2	For bit variables: bit address For byte variables: 0=lower, 8=higher byte For string variables: Length of string (max. 63 characters)
ARRAYSIZE	N	16	Number of variables in the array for index variables ATTENTION: Only the first variable is fully available. All others are only available for VBA or the Recipegroup Manager
LES_SCHR	L	1	Write-Read-Authorization 0: Not allowed to set value. 1: Allowed to set value.
MIT_ZEIT	R	1	time stamp in zenon (only if supported by the driver)
OBJEKT	N	2	Driver-specific ID number of the primitive object comprises TREIBER-OBJEKTTYP and DATENTYP
SIGMIN	Floa t	16	Non-linearized signal - minimum (signal resolution)
SIGMAX	F	16	Non-linearized signal - maximum (signal resolution)
ANZMIN	F	16	Technical value - minimum (measuring range)
ANZMAX	F	16	Technical value - maximum (measuring range)
ANZKOMMA	N	1	Number of decimal places for the display of the values (measuring range)
UPDATERATE	F	19	Update rate for mathematics variables (in sec, one decimal possible) not used for all other variables

Identification	Typ e	Field size	Comment
MEMTIEFE	N	7	Only for compatibility reasons
HDRATE	F	19	HD update rate for historical values (in sec, one decimal possible)
HDTIEFE	N	7	HD entry depth for historical values (number)
NACHSORT	R	1	HD data as postsorted values
DRRATE	F	19	Updating to the output (for zenon DDE server, in [s], one decimal possible)
HYST_PLUS	F	16	Positive hysteresis, from measuring range
HYST_MINUS	F	16	Negative hysteresis, from measuring range
PRIOR	N	16	Priority of the variable
REAMATRIZE	C	32	Allocated reaction matrix
ERSATZWERT	F	16	Substitute value, from measuring range
SOLLMIN	F	16	Minimum for set value actions, from measuring range
SOLLMAX	F	16	Maximum for set value actions, from measuring range
VOMSTANDBY	R	1	Get value from standby server; the value of the variable is not requested from the server but from the Standby Server in redundant networks
RESOURCE	C	128	Resources label. Free string for export and display in lists. The length can be limited using the MAX_LAENGE entry in **project.ini**.
ADJWVBA	R	1	Non-linear value adaption: *0*: Non-linear value adaption is used *1*: Non-linear value adaption is not used
ADJZENON	C	128	Linked VBA macro for reading the variable value for non-linear value adjustment.
ADJWVBA	C	128	ed VBA macro for writing the variable value for non-linear value adjustment.
ZWREMA	N	16	Linked counter REMA.

Identification	Typ e	Field size	Comment
MAXGRAD	N	16	Gradient overflow for counter REMA.

Attention

When importing, the driver object type and data type must be amended to the target driver in the DBF file in order for variables to be imported.

LIMIT VALUE DEFINITION

Limit definition for limit values *1* to *4*, or status *1* to *4*:

Identification	Type	Field size	Comment
AKTIV1	R	1	Limit value active (per limit value available)
GRENZWERT1	F	20	technical value or ID number of a linked variable for a dynamic limit value (see VARIABLEx) (if VARIABLEx is *1* and here it is *-1*, the existing variable linkage is not overwritten)
SCHWWERT1	F	16	Threshold value for limit value
HYSTERESE1	F	14	Is not used
BLINKEN1	R	1	Set blink attribute
BTB1	R	1	Logging in CEL
ALARM1	R	1	Alarm
DRUCKEN1	R	1	Printer output (for CEL or Alarm)
QUITTIER1	R	1	Must be acknowledged
LOESCHE1	R	1	Must be deleted
VARIABLE1	R	1	Dyn. limit value linking the limit is defined by an absolute value (see field GRENZWERTx).
FUNC1	R	1	Functions linking
ASK_FUNC1	R	1	Execution via Alarm Message List
FUNC_NR1	N	10	ID number of the linked function

Identification	Type	Field size	Comment
			(if "-1" is entered here, the existing function is not overwritten during import)
A_GRUPPE1	N	10	Alarm/Event Group
A_KLASSE1	N	10	Alarm/Event Class
MIN_MAX1	C	3	Minimum, Maximum
FARBE1	N	10	Color as Windows coding
GRENZTXT1	C	66	Limit value text
A_DELAY1	N	10	Time delay
INVISIBLE1	R	1	Invisible

Expressions in the column "Comment" refer to the expressions used in the dialog boxes for the definition of variables. For more information, see chapter Variable definition.

7.5 Communication details (Driver variables)

The driver kit implements a number of driver variables. This variables are part of the driver object type *Communication details*. These are divided into:

- Information
- Configuration
- Statistics and
- Error message

The definitions of the variables implemented in the driver kit are available in the import file **DRVVAR.DBF** and can be imported from there.
Path to file: *%ProgramData%\COPA-DATA\zenon<Versionsnummer>\PredefinedVariables*

Note: Variable names must be unique in zenon. If driver variables of the driver object type *Communication details* are to be imported from **DRVVAR.DBF** again, the variables that were imported beforehand must be renamed.

Information

Not every driver supports all driver variables of the driver object type *Communication details*.

For example:

- Variables for modem information are only supported by modem-compatible drivers.

- Driver variables for the polling cycle are only available for pure polling drivers.
- Connection-related information such as **ErrorMSG** is only supported for drivers that only edit one connection at a a time.

INFORMATION

Name from import	Type	Offset	Description
MainVersion	*UINT*	0	Main version number of the driver.
SubVersion	*UINT*	1	Sub version number of the driver.
BuildVersion	*UINT*	29	Build version number of the driver.
RTMajor	*UINT*	49	zenon main version number
RTMinor	*UINT*	50	zenon sub version number
RTSp	*UINT*	51	zenon Service Pack number
RTBuild	*UINT*	52	zenon build number
LineStateIdle	*BOOL*	24.0	TRUE, if the modem connection is idle
LineStateOffering	*BOOL*	24.1	TRUE, if a call is received
LineStateAccepted	*BOOL*	24.2	The call is accepted
LineStateDialtone	*BOOL*	24.3	Dialtone recognized
LineStateDialing	*BOOL*	24.4	Dialing active
LineStateRingBack	*BOOL*	24.5	While establishing the connection
LineStateBusy	*BOOL*	24.6	Target station is busy
LineStateSpecialInfo	*BOOL*	24.7	Special status information received
LineStateConnected	*BOOL*	24.8	Connection established
LineStateProceeding	*BOOL*	24.9	Dialing completed
LineStateOnHold	*BOOL*	24.10	Connection in hold
LineStateConferenced	*BOOL*	24.11	Connection in conference mode.
LineStateOnHoldPendConf	*BOOL*	24.12	Connection in hold for conference

Name from import	Type	Offset	Description
LineStateOnHoldPendTransfe r	*BOOL*	24.13	Connection in hold for transfer
LineStateDisconnected	*BOOL*	24.14	Connection terminated.
LineStateUnknow	*BOOL*	24.15	Connection status unknown
ModemStatus	*UDINT*	24	Current modem status
TreiberStop	*BOOL*	28	Driver stopped For *driver stop*, the variable has the value *TRUE* and an **OFF** bit. After the driver has started, the variable has the value *FALSE* and no **OFF** bit.
SimulRTState	*UDINT*	60	Informs the status of Runtime for driver simulation.
ConnectionStates	*STRING*	61	Internal connection status of the driver to the PLC. Connection statuses: *0:* Connection OK *1:* Connection failure *2:* Connection simulated Formating: *<Netzadresse>:<Verbindungszustand>;…;…;* A connection is only known after a variable has first signed in. In order for a connection to be contained in a string, a variable of this connection must be signed in once. The status of a connection is only updated if a variable of the connection is signed in. Otherwise there is no communication with the corresponding controller.

CONFIGURATION

Name from import	Type	Offset	Description
ReconnectInRead	*BOOL*	27	If TRUE, the modem is automatically reconnected for reading
ApplyCom	*BOOL*	36	Apply changes in the settings of the serial interface. Writing to this variable immediately results in the method SrvDrvVarApplyCom being called (which currently has no further function).
ApplyModem	*BOOL*	37	Apply changes in the settings of the modem. Writing this variable immediately calls the method SrvDrvVarApplyModem. This closes the current connection and opens a new one according to the settings **PhoneNumberSet** and **ModemHwAdrSet**.
PhoneNumberSet	*STRING*	38	Telephone number, that should be used
ModemHwAdrSet	*DINT*	39	Hardware address for the telephone number
GlobalUpdate	*UDINT*	3	Update time in milliseconds (ms).
BGlobalUpdaten	*BOOL*	4	TRUE, if update time is global
TreiberSimul	*BOOL*	5	TRUE, if driver in sin simulation mode
TreiberProzab	*BOOL*	6	TRUE, if the variables update list should be kept in the memory
ModemActive	*BOOL*	7	TRUE, if the modem is active for the driver
Device	*STRING*	8	Name of the serial interface or name of the modem
ComPort	*UINT*	9	Number of the serial interface.
Baudrate	*UDINT*	10	Baud rate of the serial interface.
Parity	*SINT*	11	Parity of the serial interface
ByteSize	*USINT*	14	Number of bits per character of the serial interface Value = *0* if the driver cannot establish any serial connection.

Name from import	Type	Offset	Description
StopBit	*USINT*	13	Number of stop bits of the serial interface.
Autoconnect	*BOOL*	16	TRUE, if the modem connection should be established automatically for reading/writing
PhoneNumber	*STRING*	17	Current telephone number
ModemHwAdr	*DINT*	21	Hardware address of current telephone number
RxIdleTime	*UINT*	18	Modem is disconnected, if no data transfer occurs for this time in seconds (s)
WriteTimeout	*UDINT*	19	Maximum write duration for a modem connection in milliseconds (ms).
RingCountSet	*UDINT*	20	Number of ringing tones before a call is accepted
ReCallIdleTime	*UINT*	53	Waiting time between calls in seconds (s).
ConnectTimeout	*UINT*	54	Time in seconds (s) to establish a connection.

STATISTICS

Name from import	Type	Offset	Description
MaxWriteTime	*UDINT*	31	The longest time in milliseconds (ms) that is required for writing.
MinWriteTime	*UDINT*	32	The shortest time in milliseconds (ms) that is required for writing.
MaxBlkReadTime	*UDINT*	40	Longest time in milliseconds (ms) that is required to read a data block.
MinBlkReadTime	*UDINT*	41	Shortest time in milliseconds (ms) that is required to read a data block.
WriteErrorCount	*UDINT*	33	Number of writing errors
ReadSucceedCount	*UDINT*	35	Number of successful reading attempts
MaxCycleTime	*UDINT*	22	Longest time in milliseconds (ms) required to

Name from import	Type	Offset	Description
			read all requested data.
MinCycleTime	*UDINT*	23	Shortest time in milliseconds (ms) required to read all requested data.
WriteCount	*UDINT*	26	Number of writing attempts
ReadErrorCount	*UDINT*	34	Number of reading errors
MaxUpdateTimeNormal	*UDINT*	56	Time since the last update of the priority group **Normal** in milliseconds (ms).
MaxUpdateTimeHigher	*UDINT*	57	Time since the last update of the priority group **Higher** in milliseconds (ms).
MaxUpdateTimeHigh	*UDINT*	58	Time since the last update of the priority group **High** in milliseconds (ms).
MaxUpdateTimeHighest	*UDINT*	59	Time since the last update of the priority group **Highest** in milliseconds (ms).
PokeFinish	*BOOL*	55	Goes to *1* for a query, if all current pokes were executed

ERROR MESSAGE

Name from import	Type	Offset	Description
ErrorTimeDW	*UDINT*	2	Time (in seconds since 1.1.1970), when the last error occurred.
ErrorTimeS	*STRING*	2	Time (in seconds since 1.1.1970), when the last error occurred.
RdErrPrimObj	*UDINT*	42	Number of the PrimObject, when the last reading error occurred.
RdErrStationsName	*STRING*	43	Name of the station, when the last reading error occurred.
RdErrBlockCount	*UINT*	44	Number of blocks to read when the last reading error occurred.
RdErrHwAdresse	*DINT*	45	Hardware address when the last reading error occurred.

Name from import	Type	Offset	Description
RdErrDatablockNo	*UDINT*	46	Block number when the last reading error occurred.
RdErrMarkerNo	*UDINT*	47	Marker number when the last reading error occurred.
RdErrSize	*UDINT*	48	Block size when the last reading error occurred.
DrvError	*USINT*	25	Error message as number
DrvErrorMsg	*STRING*	30	Error message as text
ErrorFile	*STRING*	15	Name of error log file

8 Driver-specific functions

The driver supports the following functions:

BLOCKWRITE

Blockwrite allows for the efficient sending of multiple set values (e.g. recipes). Variables that lie next to each other in the PLC memory will be written to with a single write telegram or combined into a few telegrams (for larger areas).

Attention: If blockwrite is activated, the write sequence of the variables does not necessarily have to match their sending sequence.

Blockwrite can be activated with an entry in the **project.ini** file:

1. Highlight the project in Project Manager
2. Press the **Ctrl**+**Alt**+**E** keyboard shortcut

   The SQL folder of zenon opens in the Windows Explorer

   C:\ProgramData\COPA-DATA\[SQL folder]\[UID]}FILES
3. Go to \zenon*system*\ .
   a) open the **project.ini** file with a text editor.
   b) Add the following entry:

      [MODBUS_ENERGY]
      BLOCKWRITE=*1*

SEQUENCE OF EVENTS

The driver contains functions for reading out the event buffer of the PLCs mentioned in chapter MODBUS_ENERGY (on page 5). This functionality can be activated via the driver dialog Settings (on page 17). The type of PLC can only be set globally for the driver.

READ PROCEDURE

The MODBUS protocol does not give the slave any possibility to report without querying the master. The controller must therefore save spontaneous events and the exact time they occurred and only transfer these once the master has queried this.

Sequence of Events (SOE) is a buffer with events (values and time stamp) that are stored in a special memory section in the PLCs. The address, the size, the format, and the procedure of reading from the buffer is manufacturer-specific as the MODBUS protocol does not have any guidelines for this.

spontaneous communication only supports the driver for events that are read from the buffer. In doing so, only these values receive a time stamp in the controller. These can therefore allocated as spontaneous events in the control system at all points (such as as alarms, in the CEL, archives).

The driver checks the occurrence of new events cyclically with the highest priority used in the driver configuration.

THE INITIAL STATUS OF THE EVENT VARIABLE

When connecting to the PLC the driver gets the initial status for events from the event buffer (PLC-specific). If an event equals a distinct address in the MODBUS register and it was engineered as *SOE register event* , it is possible for the driver to adjust the possible initial values with the respective register. If however the initial status of the event variable was read out from the register (and not from the event buffer) and the original time stamp of the event is not available in the PLC anymore, this variable receives the local PC time as time stamp and receives the status bit T_INTERN **(real time internal)**.

STATUS OF THE TCP/IP CONNECTION

The status of the TCP/IP connection to the controller can be read with the help of a *Status* object type variable. The variables have a data type, **Offset** *0* and corresponding **Net address**. The variables relate to both IP addresses if the secondary connection was defined in the driver configuration: Type: INT or UINT:

Bit	Meaning
Bit 1	Displays whether the primary connection is active (is used for communication).
Bit 2	Establishing a connection to the primary IP-address.

Bit	Meaning
Bit 3	Failed to establish a connection to the primary IP address.
Bit 5	Displays whether the secondary connection is active (is used for communication).
Bit 6	Establishing a connection to the alternative IP address.
Bit 7	Failed to establish a connection to the alternative IP address.

AREVA MICOM P125/P126/P127

After the communication has started the driver fetches all events which are available in the PLC from the buffer (Register Page 35H). After that the retrieval (with acknowledgment) of the saved event recording of the Page 36H takes place.

For register events which are not available in the buffer the initial value - if available on Page 0H - is read out from the respective MODBUS register and stamped with the local PC time.

Numbered Events which are not available in the buffer receive initial value 0 and are stamped with the local PC time.

GE MULTILIN F650

After the communication has started the driver fetches the initial status of the events from the register (address 0xF000). After that the retrievals from the ring buffer (0xFD00) take place.

The events reported by the PLC arrive with the time stamp of the PLC and also receive the status bit T_EXTERN **(real time external)**.

SCHNEIDER SEPAM

Events are created in area "SOE Numbered Events". As offset the coil bit address of the setting is stated which should be read spontaneously. Supported are addresses 1000h - 105Fh. The read out of the initial state after a new connection is supported.

THE DRIVER OBJECT TYPE OF THE EVENTS

For the events two **Driver object types** are possible: *SOE - Numbered event* and *SOE - Register event* whereas not both have a function for each PLC.

AREVA MICOM P125/P126/P127

For Areva MiCOM both object types - *Numbered event* and *Register event* - are supported.

The *SOE - Numbered event* matches an event which is only generated if a condition occurs (and not if a condition disappears). The identification of an event is defined with the help of the **Offset** setting in the variable addressing (e.g. offset 5 for event 05). If there is an event in the PLC, the associated BOOL variable will be set to 1 and then immediately set to 0 by the driver. The coming value can be evaluated by creating an alarm/CEL/archive or a function for a limit value violation.

The *Register events* match the rest of the events. The *Register events* are assigned to one or several bits in a special register of the Page 0 in the AREVA PLC (e.g. offset 20h, bit 0). This means that the PLC makes sure that the event reflects the change of a status in a register. In the variable addressing the identification of the event takes place via **Offset** and **Bit** and describes the associated data points in a Page 0 register. You can use the datatypes BOOL, USINT and UINT.

Consult the documentation of the AREVA PLC to find out about the event types and which events are supported by the AREVA PLC.

GE MULTILIN F650

For GE Multilin F650 only object type *SOE - Numbered event* is supported.

Via the driver object type*SOE - Numbered event*, you can assign a BOOL variable to every event. The identification of the event is defined via the **Offset** setting of the variables (0 - 191).

The driver object type *SOE - Register event* has no function for the GE Multilin F650 PLC.

8.1 ICE NPx800

The reading of the events is only supported for the simplified event mode of the NPx800: only On events or Wiper events but no Status events (on/off).

Event variables are created under the driver object type *SOE-Numbered events* as BOOL variables. The event parameters can be read by creating a string variable with the same driver object type and offset.

Attention: The string variable only has a valid value when the event has just been triggered. Otherwise the value of the string variable is an empty string. It behaves in the same way as a BOOL variable , i.e. just like a Wiper event. The zenon AML/CEL entry can be triggered either by the BOOL variables or by the string variable. The display of the string is only possible via dynamic limit value texts (Variablen.chm::/15311.htm).

The event number is configured with the help of the offset.

VALUE OF THE EVENT VARIABLE

The process if an event occurs:

1. The parameters are written in a possibly existing string variable.
2. The possibly existing BOOL variable is set to 1.

3. The possibly existing BOOL variable is set to 0.
4. The length of the possibly existing string variable with the parameters is set to 0.

PARAMETER STRING

In string variables, the parameter string consists of integer values that are separated by semicolons (**;**). These integer values correspond to the following data of the event:

- 1: Event data at word offset 2
- 2: Event data at word offset 7 (parameter 0)
- 3: Event data at word offset 8 (parameter 1)
- 4: Event data at word offset 9 (parameter 2)
- 5: Event data at word offset 10 (parameter 3)
- 6: Event data at word offset 11 (parameter 4)
- 7: Event data at word offset 12 (parameter 5)
- 8: Event data at word offset 13 (parameter 6)
- 9: Event data at word offset 14

PROCEDURE OF EVENTS

Procedure for an event:

- When Runtime is started or when the driver connection is restored, all events are initialized with *0*.
- If an event is received, then:
  - A string variable is filled with the **event parameters** (if these have been created)
  - The status of the **A/D** bits of the **BOOL** variable are allocated

    If, once Runtime has started or if the driver connection is reinstated, a *0* event is received, this change of value cannot be perceived.
    If a *1* event is sent twice consecutively (without a *0* event), the second event cannot be perceived.
  - The string variable reset (empty string)

The string variable can also be evaluated with the event parameters. In the first parameter, the value of the **A/D** bit is present. All events in the RT can be perceived using the string variable.

CHECKING THE EV FLAG

The driver checks the EV flag for the ICELEC alarm stack. If the value *false* is returned, the read data is ignored and no other event are read.

FILE TRANSFER DISTURBANCE RECORDS

A string variable can be created for the file transfer of **disturbance records**. It is possible to read and delete data with these.

A file is identified in the process by means of the number of the **disturbance record descriptor**.
Possible values: *1*, *2*, *3*, *4*

The records are in **COMTRADE**-format and have a serial number (*0 - 255*). The description of the records is stored in the descriptors (*1 - 4*). These contain the date, time, serial number, source and position. A maximum of 4 disturbance records can be created.

the target folder is defined in the driver configuration in the Settings (on page 17) tab.

UPLOAD AND DELETE

- Upload file: The **GET x** entry in the string variables uploads the corresponding file.
- Delete file: The **DEL x** entry in the string variables deletes the corresponding file.

x means **Disturbance record descriptor** number (*1 - 4*)

FILE NAME

The file is named according to the following schematics

A_yyyy-mm-dd_hh-mm-ss-MMM_DescriptorZ_(x-y)

Meaning:

- **A**: Hardware address of the device
- **yyyy:** Year of creation.
- **mm:** Month of creation.
- **hh:** Hour of creation.
- **mm:** Minute of creation.
- **ss:** Second of creation.
- **MMM**: Millisecond of creation.
- **Z**: serial number of the record
- **x**: Source
- **y**: Position

For example:

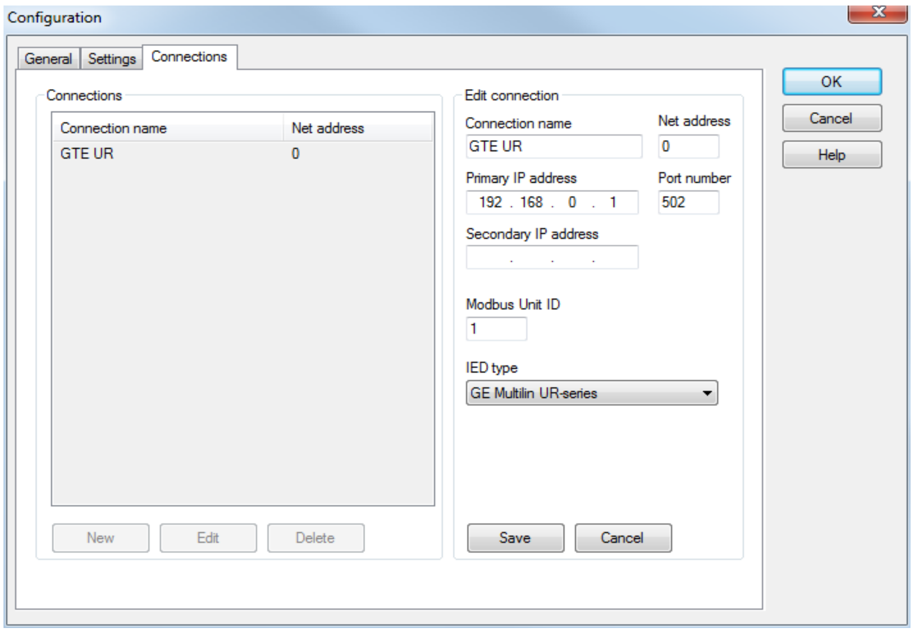
1_2009-09-07_14-01-36-690_Disturbance_109_(19-420).HDR

1_2009-09-07_14-01-36-690_Disturbance_109_(19-420).CFG

1_2009-09-07_14-01-36-690_Disturbance_109_(19-420).DAT

8.2 GE Multilin UR series

The IEC type is selected in the Connections (on page 21) tab of the driver configuration:



Configuration

General | Settings | Connections

Connections

Connection name	Net address
GTE UR	0

Edit connection

- Connection name: GTE UR
- Net address: 0
- Primary IP address: 192 . 168 . 0 . 1
- Port number: 502
- Secondary IP address: . . .
- Modbus Unit ID: 1
- IED type: GE Multilin UR-series

Click on the drop-down list in the **IEC type** section to open the selection of the IEC types. Select **GE Multilin UR series** there.

SOE SUPPORT

Via the SOE functionality of the driver events can be read from the device. For this file **EVT.TXT** is analyzed at the device. Events must be created of driver object type *SOE - Numbered event* and of data type *BOOL*. The addressing takes places via the offset setting.

A list of event numbers and their meaning can be called up via the web server of the UR series. **http://IP_Address_ of_UR_unit/FlexOperandStates.htm** (e.g.: **http://192.168.37.67/FlexOperandStates.htm**).

The *SOE - Numbered event* matches an event which is only generated if a condition occurs (and not if a condition disappears). The identification of an event is defined with the help of the **Offset** setting in the variable addressing (e.g. offset 5 for event cause 5). If there is an event in the PLC, the associated BOOL variable will be set to 1 and then immediately set to 0 by the driver. The coming value can be evaluated by creating an alarm/CEL/archive or a function for a limit value violation.

FILE TRANSFER

Via a command variable of the driver object type **File transfer** (see file transfer description for ICE devices) the file is transferred to the local computer. The command consists of: **GET** followed by a blank space and the file name: **GET EVT.TXT**

The file is copied to the file transfer folder (as defined on tab Settings (on page 17) of the configuration). Format of the file name: *Net address+underscore+file name.*
The net address is taken over from the driver/variable and is only used for saving so that the data are not overwritten for several controls.

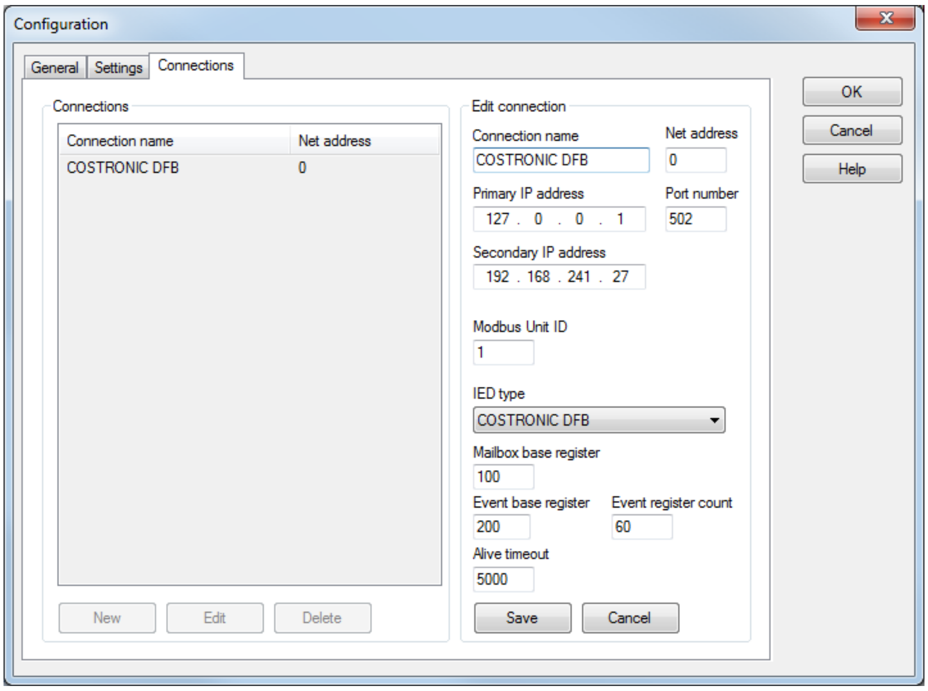
Example: **0_EVT.TXT**

8.3 COSTRONIC DFB

ALIVE TIMEOUT: DETECTION OF A PLC STOP

It is possible to detect if the PLC has stopped for COSTRONIC DFB. To do this, the counter for the event communication in the Costronics communication mailbox is monitored. If this counter does not change during a configurable waiting period, a status variable for the connection is set to *1*. the next time the counter is changed, it is set to *0* again.

Configuration is carried out using the **Alive timeout** option in the driver configuration.



Connections

Connection name	Net address
COSTRONIC DFB	0

Edit connection

Connection name: COSTRONIC DFB
Net address: 0
Primary IP address: 127 . 0 . 0 . 1
Port number: 502
Secondary IP address: 192 . 168 . 241 . 27
Modbus Unit ID: 1
IED type: COSTRONIC DFB
Mailbox base register: 100
Event base register: 200
Event register count: 60
Alive timeout: 5000

The status variable must be of the driver object type and have the **Offset** *1*. The following values are available:

Value	Meaning
0	Event mailbox has not been read yet.
1	*Alive* Counter is increased.
2	*Alive* Counter waiting time has expired.

Attention: The *Alive* counter is part of the event mailbox. If the event mailbox is not read off (no variables are active), the status is not updated. That means: In order for the *Alive* status to be monitored, at least one event variable must be active.

DRIVER-SPECIFIC COMMAND FOR UPDATE EVENT STATUS

With a driver-specific command (on page 56), it is possible to force the reading in of the current values of event-controlled values.

- Command: *UpdateEventState <HwNb>*

  Forces the driver to read all event statuses from the PLC.

Example: *UpdateEventState 1*

9 Driver command function

The zenon **Driver commands** function is to influence drivers using zenon.
You can do the following with a driver command:

- Start
- Stop
- Shift a certain driver mode
- Instigate certain actions

Note: This chapter describes standard functions that are valid for most zenon drivers.
Not all functions described here are available for every driver. For example, a driver that does not, according to the data sheet, support a modem connection also does not have any modem functions.

Attention

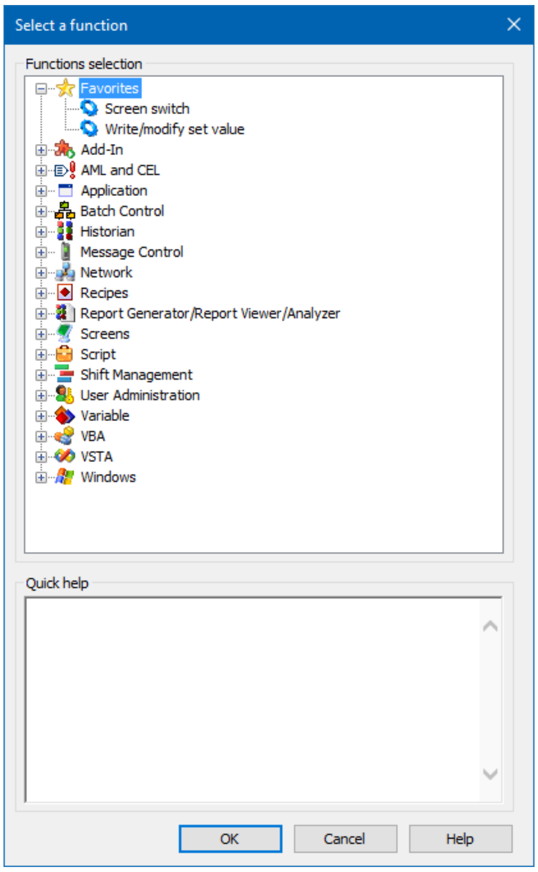
The zenon **Driver commands** function is not identical to driver commands that can be executed in the Runtime with Energy drivers!

CONFIGURATION OF THE FUNCTION

Configuration is carried out using the **Driver commands** function.
To configure the function:

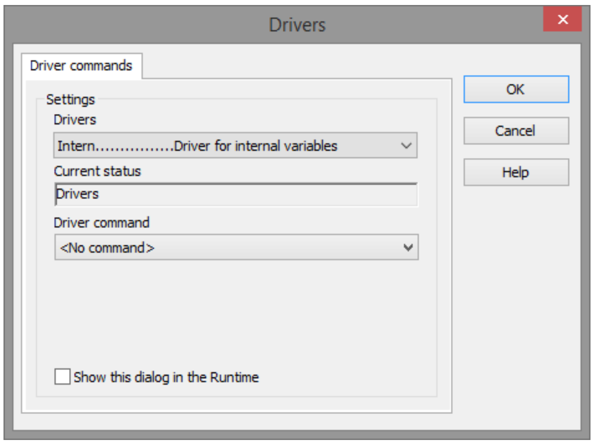
1. Create a new function in the zenon Editor.

   The dialog for selecting a function is opened



2. Navigate to the node **Variable.**
3. Select the **Driver commands** entry.

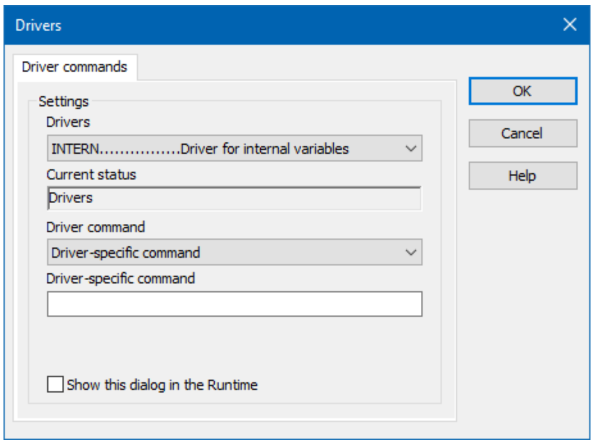
The dialog for configuration is opened



Settings
Drivers: Intern...............Driver for internal variables
Current status: Drivers
Driver command: <No command>
Show this dialog in the Runtime

4. Select the desired driver and the required command.
5. Close the dialog by clicking on **OK** and ensure that the function is executed in the Runtime. Heed the notices in the **Driver command function in the network** section.

DRIVER COMMAND DIALOG



Settings
Drivers: INTERN...............Driver for internal variables
Current status: Drivers
Driver command: Driver-specific command
Driver-specific command
Show this dialog in the Runtime

Option	Description
Driver	Selection of the driver from the drop-down list. It contains all drivers loaded in the project.
Current condition	Fixed entry that is set by the system. Has no function in the current version.
Driver command	Selection of the desired driver command from a drop-down list. For details on the configurable driver commands, see the **available driver commands** section.
Driver-specific command	Entry of a command specific to the selected driver.

Option	Description
	Note: Only available if, for the **driver command** option, the *driver-specific command* has been selected.
Show this dialog in the Runtime	Configuration of whether the configuration can be changed in the Runtime: ▸ *Active*: This dialog is opened in the Runtime before executing the function. The configuration can thus still be changed in the Runtime before execution. ▸ *Inactive*: The Editor configuration is applied in the Runtime when executing the function. Default: *inactive*

CLOSE DIALOG

Options	Description
OK	**Applies settings and closes the dialog.**
Cancel	Discards all changes and closes the dialog.
Help	Opens online help.

AVAILABLE DRIVER COMMANDS

These driver commands are available - depending on the selected driver:

Driver command	Description
<No command>	No command is sent. A command that already exists can thus be removed from a configured function.
Start driver (online mode)	Driver is reinitialized and started. **Note:** If the driver has already been started, it must be stopped. Only then can the driver be re-initialized and started.
Stop driver (offline mode)	Driver is stopped. No new data is accepted. **Note:** If the driver is in offline mode, all variables that were created for this driver receive the status *switched off* (*OFF*; Bit *20*).

Driver command	Description
Driver in simulation mode	Driver is set into simulation mode. The values of all variables of the driver are simulated by the driver. No values from the connected hardware (e.g. PLC, bus system, ...) are displayed.
Driver in hardware mode	Driver is set into hardware mode. For the variables of the driver the values from the connected hardware (e.g. PLC, bus system, ...) are displayed.
Driver-specific command	Entry of a driver-specific command. Opens input field in order to enter a command.
Activate driver write set value	Write set value to a driver is possible.
Deactivate driver write set value	Write set value to a driver is prohibited.
Establish connection with modem	Establish connection (for modem drivers) Opens the input fields for the hardware address and for the telephone number.
Disconnect from modem	Terminate connection (for modem drivers)
Driver in counting simulation mode	Driver is set into counting simulation mode. All values are initialized with *0* and incremented in the set update time by *1* each time up to the maximum value and then start at *0* again.
Driver in static simulation mode	No communication to the controller is established. All values are initialized with *0*.
Driver in programmed simulation mode	The values are calculated by a freely-programmable simulation project. The simulation project is created with the help of the zenon Logic Workbench and runs in the zenon Logic Runtime.

DRIVER COMMAND FUNCTION IN THE NETWORK

If the computer on which the **Driver commands** function is executed is part of the zenon network, further actions are also carried out:

- A special network command is sent from the computer to the project server.
  It then executes the desired action on its driver.
- In addition, the Server sends the same driver command to the project standby.
  The standby also carries out the action on its driver.

This makes sure that Server and Standby are synchronized. This only works if the Server and the Standby both have a working and independent connection to the hardware.

9.1 SwitchConnection

The driver-specific command *SwitchConnection* switches between a primary and secondary IP address.

Procedure:

1. The active IP address is closed for the **Net address** entered with the command.
2. In doing so, the **INVALID** bit is not set for the variables.
3. The connection is then established via the other IP address.
4. If the connection is not successfully established, the **INVALID** bit is set and a switch back to the active connection is made.

Note: The driver command is taken into account if the connection to an IP address has been established successfully. The command is ignored if there is no connection or a connection is currently being established.

Syntax:

- *SwitchConnection <NetAddr>*
  <NetAddr> Stands for the number of the **Net address**.

Example:

- **SwitchConnection 1**
  Switches between primary and secondary address for the connection to net address *1*.

10 Error analysis

Should there be communication problems, this chapter will assist you in finding out the error.

10.1 Analysis tool

All zenon modules such as Editor, Runtime, drivers, etc. write messages to a joint log file. To display them correctly and clearly, use the Diagnosis Viewer (main.chm::/12464.htm) program that was also installed with zenon. You can find it under **Start/All programs/zenon/Tools 8.10 -> Diagviewer.**

zenon driver log all errors in the LOG files.LOG files are text files with a special structure. The default folder for the LOG files is subfolder **LOG** in the folder **ProgramData**. For example:

%ProgramData%\COPA-DATA\LOG.

Attention: With the default settings, a driver only logs error information. With the Diagnosis Viewer you can enhance the diagnosis level for most of the drivers to "Debug" and "Deep Debug". With this the driver also logs all other important tasks and events.

In the Diagnosis Viewer you can also:

- Follow newly-created entries in real time
- customize the logging settings
- change the folder in which the LOG files are saved

Note:

1. The Diagnosis Viewer displays all entries in UTC (coordinated world time) and not in local time.
2. The Diagnosis Viewer does not display all columns of a LOG file per default. To display more columns activate property **Add all columns with entry** in the context menu of the column header.
3. If you only use **Error-Logging**, the problem description is in the column **Error text**. For other diagnosis level the description is in the column **General text**.
4. For communication problems many drivers also log error numbers which the PLC assigns to them. They are displayed in **Error text** or **Error code** or **Driver error parameter** (**1** and **2)**. Hints on the meaning of error codes can be found in the driver documentation and the protocol/PLC description.
5. At the end of your test set back the diagnosis level from **Debug** or **Deep Debug**. At **Debug** and **Deep Debug** there are a great deal of data for logging which are saved to the hard drive and which can influence your system performance. They are still logged even after you close the Diagnosis Viewer.

> **Attention**
>
> In Windows CE errors are not logged per default due to performance reasons.

You can find further information on the Diagnosis Viewer in the Diagnose Viewer (main.chm::/12464.htm) manual.

10.2 Check list

Problem	Diagnostics	Reason
Values can be read or written by the controller.	The controller can be contacted by 'pinging'?	▸ The controller is not connected to the power supply or the network. ▸ The PC is not connected to the network. ▸ The controller is connected but is in

Problem	Diagnostics	Reason
		a different subnetwork and the network gateway is not entered in the controller or the subnetmask is not set correctly. ▸ Is the firewall activated? Port 502 is used for communication as standard; add it to the exceptions. Enter accordingly for individual port numbers.
	The controller can be contacted by 'pinging'?	▸ The communication parameters are not set correctly? ▸ The port must be set according to the configuration of the controller (Modbus Slave). ▸ The network address in the addressing of the variable does not correspond to the network address of the connection in the driver. Attention: Network address 0 must not be used. Address 0 is reserved as a broadcast address in Modbus. ▸ The driver configuration file was not transferred to the target computer?
	The PLC is connected serially	▸ The controller is not connected to the power supply or the bus system. ▸ The serial cable is not connected to the correct interface (COM1...64), or the interface was set incorrectly. ▸ The serial interface is blocked by another application. ▸ The network address in the addressing of the variable does not correspond to the Modbus address (device address). ▸ The cable is assigned incorrectly or defective.
Certain values cannot be read or	Has an analysis with the Diagnosis Viewer been carried	▸ See the following chapter: Error numbers

Problem	Diagnostics	Reason
written by the controller.	out to see which errors have occurred? See Analysis tool (on page 61) chapter.	▸ Are the variables correctly addressed? ▸ Are the correct object types used in the variable? The object types determine the function code to be used in the Modbus telegram.
Incorrect values are displayed.	Has an analysis with the Diagnosis Viewer been carried out to see which errors have occurred? See Analysis tool (on page 61) chapter.	▸ Are the variables correctly addressed? ▸ Are the correct data types used? ▸ Is the value calculation correct?